GEM DOS

Programmer's Guide

CONTENTS OF THIS GUIDE

Figures

Tables

FOREWORD

GEM DOS™ is a single-user operating system designed for the Motorola MC68000™ or compatible 68000 microprocessor. GEM DOS requires the following minimum system configuration:

* 68000 microprocessor
* Bit-mapped video controller card and a display with 320 by 200 resolution
* 128K bytes of Random Access Memory (RAM)
* 160K bytes of Read-Only Memory (ROM)
* An optical or track mouse or other pointing device

The following configuration represents a system that can exercise the full capabilities of GEM DOS:

* 68010, 68020, or 68070 advanced microprocessor
* High-resolution, color bit-mapped display and controller (640 by 400 pixel resolution, barrel-shifter hardware for very high speed screen refresh)
* 256K bytes of RAM
* 192K bytes of ROM
* Mouse
* Disk drives and controllers
* Optical drives and controllers
* Printer and communication ports

This guide describes the programming interface to GEM DOS and is part of a documentation set that also includes these manuals:

* The GEM DOS User's Guide

* The C Language Programming Guide for GEM DOS (supplement to the C Language Programming Guide for CP/M-68K..)

* The Mince User's Guide, prepared by Mark of the Unicorn, Inc.

* The Kermit User's Guide prepared by Columbia University Center for Computing Activities

* The GEM DOS Supplement to the Kermit User's Guide

This guide assumes you are an experienced programmer familiar with the Motorola 68000 assembly language and the C programming language. If you are not familiar with GEM DOS, refer to the manuals listed above, especially the GEM DOS User's Guide. If you are not familiar with the Motorola 68000 assembly language, refer to the following Motorola manuals:

* 16-Bit Microprocessor User's Manual, third edition MC68000UM(AD3)

* MC68000 Resident Structured Assembler Reference Manual M68KMASM(D4)

The following is a summary of the contents of this guide:

* Section 1 discusses the GEM DOS system architecture, file specification, and terminology.

* Section 2 describes the programming loading procedure, transient program environment, and how to exit a transient program.

* Section 3 describes the GEM DOS comand file (transient program) format.

* Section 4 lists the system function call categories and describes the system functions and their entry parameters and return codes.

* Section 5 describes the GEM DOS assembler.

* Section 6 describes the GEM DOS linkers.

* Section 7 describes the GEM DOS archive utility.

* Section 8 describes the DUMP, SIZE68, and SENDC68 utilities.

* Section 9 describes the SID-68K™ debugger.

* Appendix A lists the BIOS functions.

* Appendix B shows the format of the GEM DOS base page.

* Appendix C contains a decimal-ASCII-hexadecimal conversion table.

* Appendix D lists the error messages returned by the GEM DOS programming utilities.

* Appendix E describes the Motorola S-Record format.

INTRODUCTION TO GEM DOS

1.1 SYSTEM ARCHITECTURE

Table 1-1 describes the three modules that comprise GEM DOS.

Table 1-1. GEM DOS System Modules

Module
Console Command Processor (CCP)
Basic Disk Operating System (BDOS)
Basic I/O System (BIOS)

1.2 TRANSIENT PROGRAMS

The Transient Program Area (TPA) consists of all the contiguous memory space not occupied by the operating system when GEM DOS is resident in memory. GEM DOS loads executable command files from disk into the TPA. Command files are also called transient programs because they are not memory resident nor configured within the operating system. Section 3 describes the format of a GEM DOS command file.

Section 2, "The CCP and Transient Programs," describes how

Section 2, "The CCP and Transient Programs," describes how GEM DOS formats the TPA for programs it loads into memory for execution.

1.3 PROGRAMMING TOOLS

Table 1-2 lists the assembly language programming tools provided with GEM DOS. These tools are described in Sections 5 through 9 of this guide.

Table 1-2. GEM DOS Programming Tools

Program	Purpose
AR68	Creates and modifies object module libraries
AS68	Processes assembly language source modules to create relocatable object module library files
DUMP	Displays files in hexadecimal and ASCII notation
LINK68™	A linker that combines assembled or compiled object modules with other modules from a runtime library
LO68	Another linker
NM68	Prints the symbol table of an object or command file
RELOC	Converts relocatable object modules from one format to another
SENDC68	Converts a command file to the Motorola S-record format
SID-68K™	Tests and debugs programs
SIZE68	Displays the size of each program segment within one or more command files
XREF	Creates a table of the symbols defined and referenced by object files

Use AS68 to assemble the file and either LINK68 or LO68 to link it. AR68 creates library modules that can be linked with LINK68 or LO68. Both of the linkers use and create the same file formats. However, if you are not linking with runtime modules, LO68 provides better performance.

NM68 is described in Section 3.

GEM DOS also supports a C language compiler for the 68000 microprocessor. See the C Language Programming Guide for GEM DOS.

1.4 GEM DOS FILE SPECIFICATION

The GEM DOS file specification consists of four fields: a one-character drive designation delimited by a colon (d:), a directory path in which directory names are delimited by backslashes (\), a one- to eight-character filename, and an optional one- to three-character filetype separated from the filename by a period (.). Directories and files have the same naming conventions. The format of a file specification is shown below.
Format:

```
d:\path\filename.typ
```

Example:

```
A:\LIBRARY\OBJS\FLOATER.OBJ
```

The drive designation, path, and filetype fields are optional. The colon, backslash, and period delimiters are required only when you include these fields in the file specification.

Drive designation values range from A through P when the BIOS implementation supports 16 drives, the maximum number that GEM DOS can support. See Section 4.5, "File Management Functions."

The characters in the filename and filetype fields cannot contain delimiters (the colon, backslash, and period). The CCP automatically converts a command line and its file specifications (if any) to uppercase internally before parsing them.

Table 1-3 lists some additional characters you should avoid using in your file specifications. These characters are reserved because several GEM DOS built-in commands and utilities have special uses for them.

Table 1-3. Delimiter Characters

Character
[]
()
<>
=
*
&
'
!
|
?
/
\
#
.
:
;
+
-

1.5 WILDCARDS

GEM DOS supports two wildcards, the question mark (?) and the asterisk(*). Several utilities and BDOS functions allow you to specify wildcards in a file specification to perform the operation or function on one or more files.

The ? wildcard matches any one character in the character position occupied by this wildcard. For example, the file specification WH?TMAN.OBJ indicates that the third letter of the filename can be any alphanumeric character if the remainder of the file specification matches. Thus, the ? wildcard matches exactly one character position.

The * wildcard matches one or more characters in the field or remainder of a field that this wildcard occupies. GEM DOS internally pads the field or remaining portion of the field occupied by the * wildcard with ? wildcards before searching for a match. For example, GEM DOS converts the file specification G*.BAT to G???????.BAT before searching for a matching file specification. Thus, any file that starts with the letter G and has a filetype of BAT matches this file specification.

For details on wildcard support by specific BDOS functions, refer to the description of the function in Section 4 of this guide. For additional details on these wildcards and support by GEM DOS utilities, refer to the GEM DOS User's Guide and Sections 5 through 9 of this guide.

1.6 GEM DOS TERMINOLOGY

Table 1-4 list the terminology used throughout this guide to describe GEM DOS values and program components.

Table 1-4. GEM DOS Terminology

Term	Meaning
Nibble	4-bit value
Byte	8-bit value
Word	16-bit value
Longword	Signed 32-bit value
Address	32-bit value that specifies a storage location
Offset	A fixed displacement that references a storage location, other data source, or destination
Text Segment	The section of a program that contains the program instructions
Data Segment	The section of a program that contains initialized data
bss	The block storage segment is a section of a program that contains uninitialized data

Table 1-5 describes conventions used in this guide.

Table 1-5. GEM DOS Programmer's Guide Conventions

Convention	Meaning
[]	Square brackets in a command line enclose optional parameters.
nH	The capital letter H follows numeric values that are represented in hexadecimal notation.
(n)	BDOS function numbers are enclosed in parentheses when they appear in text.
Ctrl-X	The mnemonic Ctrl-X indicates that you are to hold down the key labeled Ctrl while you press the key represented by the variable X.

End of Section 1

THE CCP AND TRANSIENT PROGRAMS

This section discusses the Console Command Processor (CCP), built-in and transient commands, loading and exiting transient programs, and the GEM DOS program execution model.

2.1 CCP BUILT-IN AND EXTERNAL COMMANDS

After an initial cold start, GEM DOS displays a sign-on message at the console. The drive containing the system disk is logged in automatically. GEM DOS displays its "(a)" prompt to indicate the currently logged in drive, and that the operating system is ready to receive a command line from the console.

In response to the prompt, a user types the filename of a batch or command file with its command tail, if required. GEM DOS supports two types of commands, built-in and transient. Built-in commands are part of the CCP. Transient commands are loaded in the TPA and do not reside in memory allocated to GEM DOS. The GEM DOS User's Guide describes the built-in and transient commands that GEM DOS supports.

A transient command is a machine-readable program file that GEM DOS loads from disk into memory for execution. Section 3 describes the format of transient command files.

When the user enters a command line, the CCP parses it and tries to execute the file specified. The CCP asssumes a file is a command file when it has the filetype BAT. When the user specifies only the filename, the CCP searches the current directory and drive for a file with a matching filename and a filetype of BAT. If the CCP does not locate a batch file to match the filename specified, it searches for a command file whose filetype is PRG.

2.2 LOADING A PROGRAM

Either the CCP or a transient program can load a program into memory with the BDOS Load and Execute a Process Function (4B) described in Section 4.7. After the program is loaded, the TPA contains the program segments (text, data, and bss), a user stack, and a base page. A base page exists for each program loaded into memory. The base page is a 256-byte data structure that defines a program's operating environment. The base page is initialized by the Load and Execute a Process (4B) function described in Section 4.7. GEM DOS allocates 256 bytes (minimum) for the program's user stack.

2.2.1 The TPA and Base Page Initialization

The format of the Transient Program Area (TPA) that GEM DOS establishes for an application program is shown in Figure 2-1.

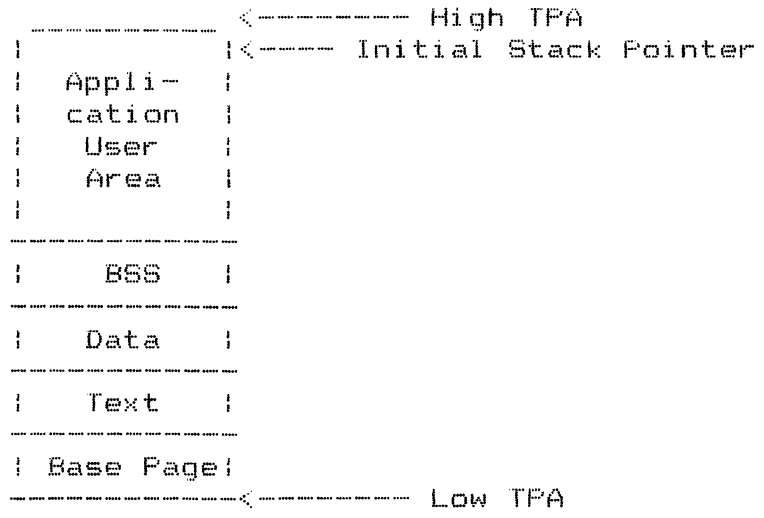


Figure 2-1. GEM DOS TPA Format

The initial stack pointer is directed at the top end of the TPA. The base page contains pointers as described in Section 2.3. To insure maximum compatibility with future releases of GEM DOS, it is recommended that memory not required by your program be freed for use by other processes. The top of the stack should be moved down to an acceptable value adjacent to or within the bss, and an M_SHRINK (4A) function should be performed. If additional memory is needed later, perform an M_FREE (49) or M_ALLOC (48) function to determine the amount of available memory (specify nbytes = -1 for the M_ALLOC call); follow this by another call to M_ALLOC to have a portion of that memory allocated. These operations reformat the TPA as shown in Figure 2-2.

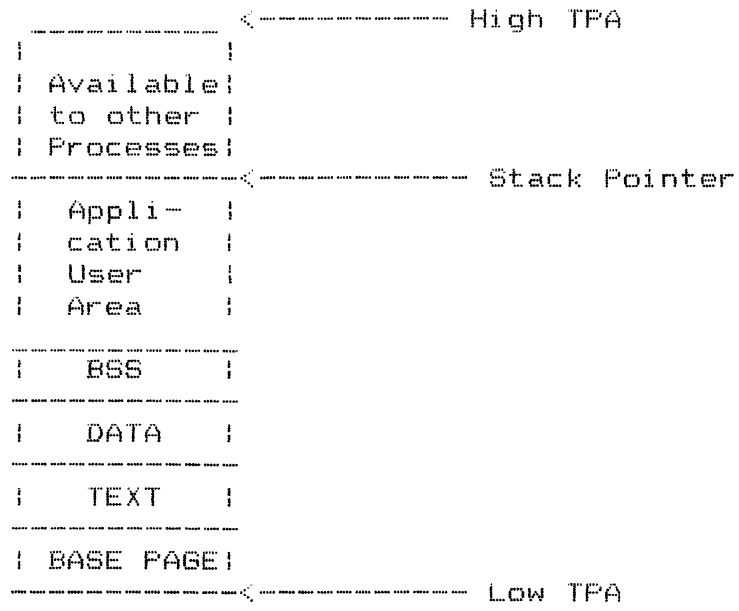


Figure 2-2. Adjusted TPA

2.3 BASE PAGE FORMAT

Table 2-1 lists the byte offset, length, value, and briefly describes the fields in the base page that Load and Execute a Process (4B) initializes for transient programs.

Table 2-1. Base Page Format

Base Page Offset	Length	Value	Description
00	4	p_lowtpa	Base address of the TPA
04	4	p_hitpa	High address of the TPA + 1
08	4	p_tbase	Base address of text (code)
0C	4	p_tlen	Length (in bytes) of text (code)
10	4	p_dbase	Base address of initialized data
14	4	p_dlen	Length (in bytes) of data
18	4	p_bbase	Base address of BSS uninitialized data
1C	4	p_blen	Length of BSS uninitialized data
2C	4	p_env	Pointer to the environment string
x80	x80	p_cmdlin	Command line image

Note: All other base page fields are reserved.

2.4 EXITING TRANSIENT PROGRAMS

GEM DOS supports two ways of exiting a transient program and returning control to the CCP:

* Interactively, when the user types Ctrl-C at the console

* A programmed return to the CCP through:

  -- Program Terminate (00)
  -- Terminate and Stay Resident (31)
  -- Terminate Process (4C)

Typing Ctrl-C from the console returns control to the CCP only if the program uses any of the following functions:

* Read Character from Standard Input Device (01)
* Write Character to Standard Output Device (02)
* Read Character from Standard Input, No Echo (08)
* Write String to Standard Output Device (09)
* Read Edited String from Standard Input Device (0A)
* Check Status of Standard Output Device (10)

Ctrl-C terminates execution of the main program and any additional programs loaded beyond the CCP level.

End of Section 2

GEM DOS COMMAND FILE FORMAT

The GEM DOS command file format is the output of the RELMOD format conversion utility. RELMOD accepts a CP/M-68K.. format relocatable output file generated by LINK68 or LO68, the GEM DOS linkers, and converts it to a GEM DOS relocatable file. RELMOD, LINK68, and LO68 are described in Section 6.

Command files consist of a file header followed by the text and data segments. Optionally, a command file contains a symbol table and relocation information. These components are shown in Figure 3-1 and described in the following sections.

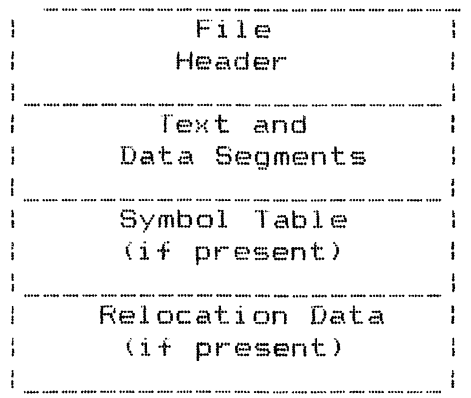


Figure 3-1. GEM DOS Command File Layout

3.1 THE HEADER AND PROGRAM SEGMENTS

The figure 3-2 shows the format of the header for non-segmented (contiguous) executable files.

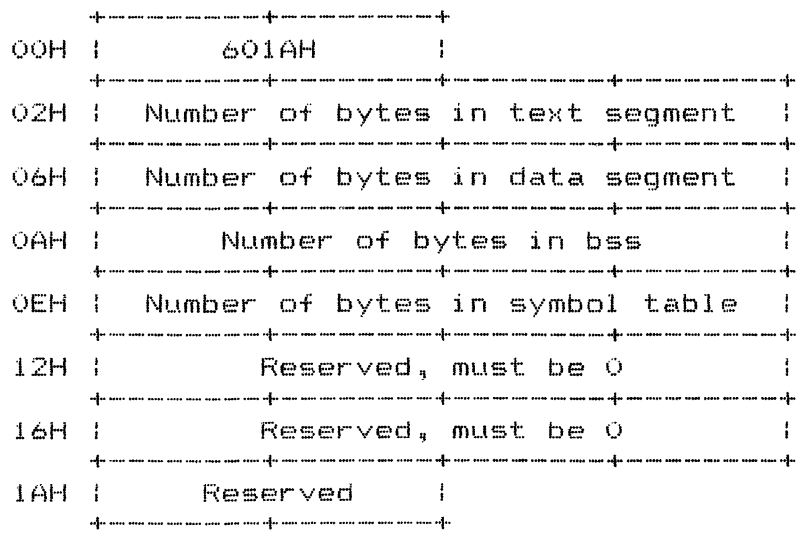


Figure 3-2. GEM DOS Command File Header

The 14-word, 601AH-type header denotes a relocatable file that contains contiguous text, data, and block storage segments. Note that GEM DOS does not support programs with non-contiguous segments.

The fields at offsets 12H, 16H, and 1AH are reserved and must be zero. The text segment image follows the field at offset 1AH.

The text segment contains the executable instruction code. The data segment contains the initialized data variables. The text and data segments must be relocatable.

The inclusion of the symbol table and relocation information in the command file is controlled by the GEM DOS linkers, described in Section 6.

3.2 SYMBOL TABLE

The symbol table defines the symbols referenced by a command file. Each entry in the symbol table is associated with an index that indicates its entry number. Entries are numbered sequentially starting with zero. As shown in Figure 3-3, each symbol table entry is composed of seven words that describe the symbol's name, type, and value.

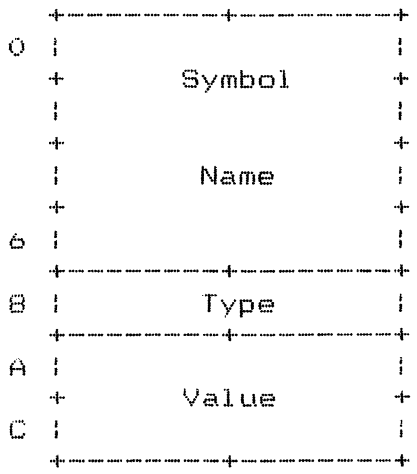


Figure 3-3. Symbol Table Entry

Table 3-1 describes the fields of a symbol table entry.

Table 3-1. Symbol Table Entry Fields

Field	Definition
Name	Symbol name, null-padded right if less than eight characters.
Type	Symbol type as indicated by the following values: 0100H: bss-based relocatable 0200H: text-based relocatable 0400H: data-based relocatable 0800H: external reference 1000H: equated register 2000H: global 4000H: equated 8000H: defined
Value	Symbol value where the value can be an address, register number, value of an expression, and so forth. Note that the linkers interpret the value when the symbol is external (0800H) as the size of a common region.

3.3 PRINTING THE SYMBOL TABLE

You can print the symbol table of an object or command file with the NM68 utility. Use the following command line to invoke NM68:

```
(a)NM filename
```

NM68 operates on object or command files. NM68 does not sort the symbols; rather, it prints them in the order in which they appear in the file. Each symbol name is printed, followed by its value and one or more of the following type-descriptors:

* equ (equated)
* global
* equreg (equated register)
* external
* data
* text
* bss
* abs (absolute)

You can use the I/O redirection facility to store NM68 output in a file. To redirect NM68 output to a file, specify a greater-than-sign (>) and a file specification following the filename for which NM68 is to print the symbol table.

3.4 RELOCATION INFORMATION

The relocation information indicates the offset between longwords to be relocated. The offset from the beginning of the text segment to the first longword that requires relocation is stored as a longword following the symbol table. Following that, bytes are used to store succeeding offsets. If the offset between two longwords to be relocated is greater than 254 bytes, the relocation information has a byte of 1 for each multiple of 254 bytes, plus a byte whose value indicates any remainder. A zero offset value indicates the end of relocation information. Note that GEM DOS does not support 16-bit relocatable objects.

See Figure 3-3. Table 3-2 lists the relocation offset byte values and their meanings.

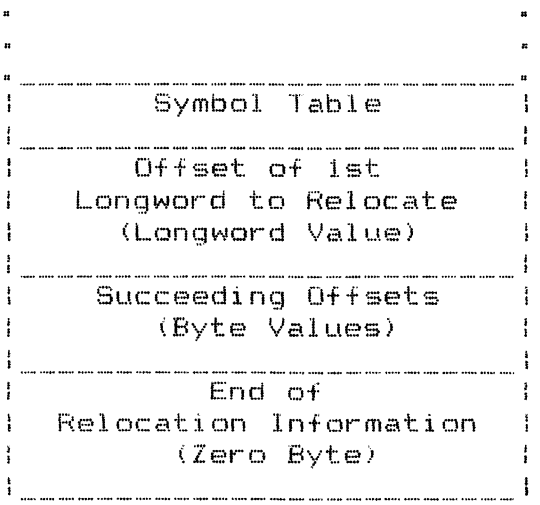


Figure 3-4. Location and Format of Relocation Information

Table 3-2. Relocation Offset Byte Table

Byte Value	
0	
1	
2 through 254	
255	*

* Plus 254 times the number of "1" bytes that preceded this byte.

All relocation information is relative to the beginning load address (start of text).

End of Section 3

SYSTEM FUNCTION CALLS

This section describes the GEM DOS system function calls, their entry parameters, return values, and error codes.

4.1 INTRODUCTION

The GEM DOS system calls are divided into the following categories: Character Device I/O, File Management, Drive and Directory Management, Program Control, Memory Management, and Time. Table 4-1 lists each GEM DOS system call according to these categories. Also listed in Table 4-1 are references to the sections of this chapter that describe each call in detail.

Table 4-1. GEM DOS System Call Categories

Number
Character Device I/O, Section 4.4
01
02
03
04
05
06
07
08
09
0A
0B
10
11
12
13
File Management, Section 4.5
1A
2F
3C
3D
3E
3F
40
41
42
43
44
45
46
4E
4F
56
57

Table 4-1. (continued)

	Number
Drive and Directory Management, Section 4.6	
	0E
	19
	36
	39
	3A
	3B
	47
Program Control, Section 4.7	
	00
	25
	30
	31
	35
	4B
	4C
Memory Management, Section 4.8	
	48
	49
	4A
Time, Section 4.9	
	2A
	2B
	2C
	2D

4.2 FUNCTION CALL PARAMETERS

The GEM DOS parameter-passing convention supports function call references from C and other high-level languages.

A function call reference from a high-level language constructs a parameter block that contains the number of the function call, any necessary parameters, and other relevant data (such as a drive code or file handle) and performs a TRAP #1 instruction. GEM DOS then processes the reference and executes the appropriate actions.

The general format of a GEM DOS function call parameter block is shown in Figure 4-1.

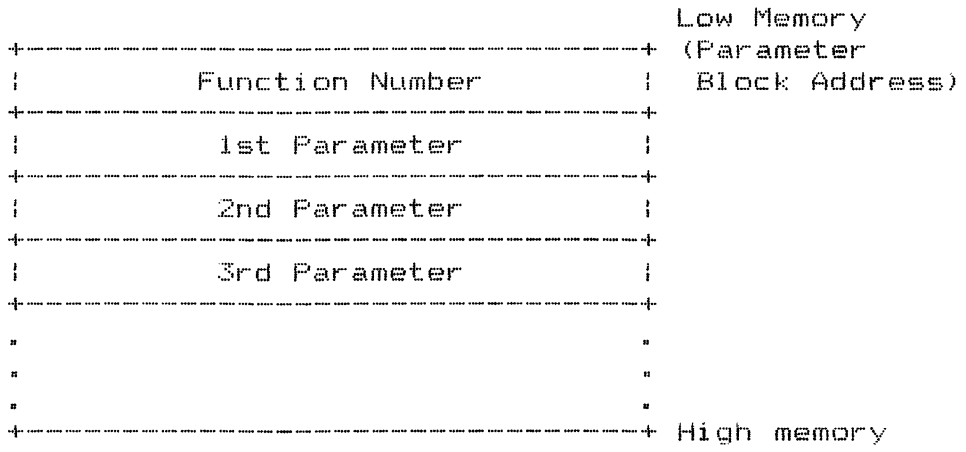


Figure 4-1. Function Call Parameter Block Format

The positions of the parameters and the function number in Figure 4-1 conform to the order in which they are placed on the stack by the C language. Specifically, the C code:

```
CALL (functionnumber, P1, P2, P3, ... Pn)
```

loads the Pn parameter into the parameter block first, followed by P3 through P1, until the function number is the last entry in the parameter block.

4.2.1 GEM DOS File and Device Handles

F_CREATE (3C), F_OPEN (3D), and F_DUP (45) return a 16-bit binary identifier value in register D0.L. This identifier is called a "handle." The handle is used to later identify files or devices that have been opened or created with the system call that originally returned it. Although a handle is 16 bits long, your program should test the entire long word value returned in D0.L to determine if the value represents an error code or a valid handle.

Table 4-2 lists the handles that identify devices.

Table 4-2. GEM DOS Device Handles

Handle Value	Device
4	Standard list device Output can be redirected.
3	Standard AUX device I/O can be redirected.
2	Standard error device (stdErr) I/O can be redirected.
1	Standard output device (stdOut) Output can redirected at the command line.
0	Standard input device (stdIn) Input can be redirected at the command line.

Your program can use these handles without previously opening them. Your program can redirect I/O for these devices with F_FORCE (46).

Your program can open the following devices by specifying a device name with the F_OPEN (3D) function:

* The console device, CON:
* The auxiliary device, AUX:
* The list device, PRN:
* The bit bucket, NUL:

Note that I/O for the devices opened by name cannot be redirected. See the descriptions of F_OPEN (3D) and F_FORCE (46) in Section 4.5.

4.2.2 Return Values

GEM DOS function calls return all byte, word, and long data in register D0 by default. Those function calls that return more information place the address of an information return buffer in D0 or fill in a caller-specified buffer. The individual function call descriptions explain how data returns to the caller.

4.2.3 Error Return Codes

The GEM DOS function calls place a negative number in D0 to indicate an error return. See the individual function call descriptions for more complete information.

Table 4-3 lists the error codes returned by the GEM DOS function calls in register D0.L. Also listed in Table 4-3 are the numbers of the PC DOS function call error codes that correspond to the GEM DOS error codes.

Table 4-3. BDOS Error Codes

GEM DOS Return	PC DOS Equiv.	Code	Error Report
-32L	1	EINVFN	Invalid function
-33L	2	EFILNF	File not found
-34L	3	EPTHNF	Path not found
-35L	4	ENHNDL	No handles left (too many open f
-36L	5	EACCDN	Access denied
-37L	6	EIHNDL	Invalid handle
-39L	8	ENSMEM	Insufficient memory
-40L	9	EIMBA	Invalid memory block address
-46L	15	EDRIVE	Invalid drive specified
-49L	18	ENMFIL	No more files
-64L		ERANGE	Range error
-65L		EINTRN	Internal error
-66L		EPLFMT	Invalid program load format
-67L		EGSBF	Setblock failure (due to growth restrictions)

4.3 GEM DOS SYSTEM CALL SUMMARY

Table 4-4 lists the hexadecimal number and mnemonic of each GEM DOS system function call. The table includes the parameters a process must pass to the system call and the values the system returns to the calling process.

Table 4-4. GEM DOS System Call Summary

Hex	Mnemonic	Input Parameters
0	P_TERMO	none
1	C_CONIN	none
2	C_CONOUT	Character
3	C_AUXIN	none
4	C_AUXOUT	Character
5	C_PRNOUT	Character
6	C_RAWIO	FF/Character
7	C_RAWCIN	none
8	C_NECIN	none
9	C_CONWS	*Character String
A	C_CONRS	*Input Buffer
B	C_CONIS	none
E	D_SETDRV	Drive Code
10	C_CONOS	none
11	C_PRNOS	none
12	C_AUXIS	none
13	C_AUXOS	none
19	D_GETDRV	none
1A	F_SETDTA	DTA (Long Value)

Table 4-4. (continued)

Hex	Mnemonic	Input Parameters
25	S_SETVEC Address	Vector#
2A	T_GETDATE	none Bits 0-4 = day (1-31) Bits 5-8 = month (1-12) Bits 9-15 = year (0-119)
2B	T_SETDATE Bits 0-4 = (1-31) Bits 5-8 = (1-12) Bits 9-15 = (0-119)	Date Word:
2C	T_GETTIME	none Bits 0-4 = 2-sec increments Bits 5-10 = minutes Bits 10-15 = hours
2D	T_SETTIME Bits 0-4 = 2-sec increments Bits 5-10 = minutes Bits 10-15 = hours	Time Word (binary):
2F	F_GETDTA	none
30	S_VERSION	none
31	P_TERMRES Exit Code	Memory Size
35	S_GETVEC	Vector#
36	D_FREE Drive Code	*Return Buffer # of Clusters Bytes/Sectors Sects/Clusters D0.L = Error Code
39	D_CREATE	*Path String
3A	D_DELETE	*Path String

Table 4-4. (continued)

Hex	Mnemonic	Input Parameters
3B	D_SETPATH	*Path String
3C	F_CREATE Attribute	*Path String
3D	F_OPEN Access Mode	*Path String
3E	F_CLOSE	Handle
3F	F_READ # of Bytes *Buffer	Handle Error Code
40	F_WRITE # of Bytes *Buffer	Handle Error Code
41	F_DELETE	*Path String
42	F_SEEK Handle Mode Value	Offset Error Code
43	F_ATTRIB Mode Value Attribute	*Path String Error Code
44	F_IOCTL Handle or Drive Code Byte Count or Bit Map *Buffer	Subfunction # Bit Map of Device Information # of Bytes Read/Written I/O Status Media Type
45	F_DUP	Handle Error Code
46	F_FORCE Duplicate Handle	Standard Handle

Table 4-4 (continued)

Hex	Mnemonic	Input Parameters
47	D_GETPATH Drive Code	*Path Buffer
48	M_ALLOC (for # of available bytes)	# of Bytes or -1L 0 on Failure
49	M_FREE	Memory Block Addr
4A	M_SHRINK # of bytes to keep	Memory Block Addr
4B	P_EXEC *Path String *Command Tail *Environment Strings	Load Value Base Page Addr Error Code
4C	P_TERM	Status Code
4E	F_SFIRST Attribute	*Path String
4F	F_SNEXT	none
56	F_RENAME *Path String (New Name)	*Path String
57	F_DATIME Buffer Handle Get/Set Flag	*Time/Date Buffer

Conventions used in Table 4-4:

*	=	Pointer to
#	=	Number
Addr	=	Address
DTA	=	Disk Transfer Address
Handle	=	File or Device Handle
Path String	=	directory path, and filename

4.4 CHARACTER DEVICE I/O FUNCTIONS

Table 4-5 lists the number, mnemonic, and name of the GEM DOS system function described in this section.

Table 4-5. Character Device I/O Functions

Hex	Mnemonic
01	C_CONIN
02	C_CONOUT
03	C_AUXIN
04	C_AUXOUT
05	C_PRNOUT
06	C_RAWIO
07	C_RAWCIN
08	C_NECIN
09	C_CONWS
0A	C_CONRS
0B	C_CONIS
10	C_CONOS
11	C_PRNOS
12	C_AUXIS
13	C_AUXOS

Function 01: Read Character from Standard Input Device

C Interface:

```
     LONG
c_conin ()
{
}
```

Parameter Block:

0	function code 01

Return Values:
Register D0.L:

(in least significant byte)

Description

C_CONIN reads a character from the standard input device and echoes it to the standard output device.

If a GEM... VDI compatible console is the standard input device, C_CONIN places the console scan code in the low byte of the high word returned in D0. Otherwise, this byte is set to 00. See Figure 4-2.

If C_CONIN reads a Ctrl-C character, it flushes the type-ahead buffer and executes a warm boot operation.

Hb Hw	Lb Hw	Hb Lw	Lb Lw
00	code or 00	00	char

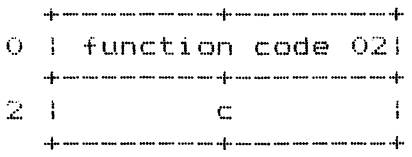
Figure 4-2. D0 Contents Returned from C_CONIN

Function 02: Write Character to Standard Ouput Device

C Interface:

```
        VOID
c_conout (c)
    WORD     c;
{
}
```

Parameter Block:



Parameters:

c	Character to be displayed in lower 8 bits, upper 8 bits should be set to zero

Description: C_CONOUT displays the character contained in the lower 8 bits of the c parameter on the standard output device. To ensure compatibility with future extensions to the 16-bit character set, the upper 8 bits of c should be zero.

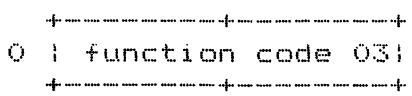
C_CONOUT translates tab characters into columns of eight characters. This function also checks for Ctrl-S (scroll stop), Ctrl-Q (strat scroll), and Ctrl-P (printer switch) and processes Ctrl-C, which aborts the operation and warm boots the system.

Function 03: Read Character from Standard Auxiliary Device

C Interface:

```
        LONG
c_auxin ()
{
}
```

Parameter Block:



Return Values:
Register D0.L:

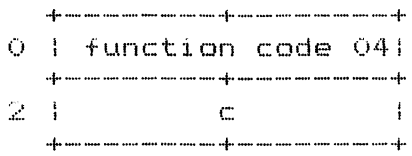
Description: C_AUXIN receives a character from the auxiliary port and returns it in D0.L.

Function 04: Write Character to Standard Auxiliary Device

C Interface:

```
  VOID
c_auxout (c)
  WORD c;
{
}
```

Parameter Block:



Parameters:

c	Character to be sent to the auxiliary port in lower 8 bits, upper 8 bits should be set to zero

Description:

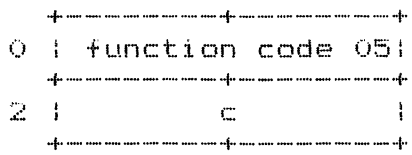
C_AUXOUT sends the character contained in the lower 8 bits of the c parameter to the auxiliary port. To ensure compatibility with future extensions to the 16-bit character set, the upper 8 bits of c should be zero.

Function 05: Write Character to Standard Print Device

C Interface:

```
  VOID
c_prnout (c)
  WORD c;
{
}
```

Parameter Block:



Parameters:

c	Character to be sent to the standard printer device in lower 8 bits, upper 8 bits should be set to zero

Description:

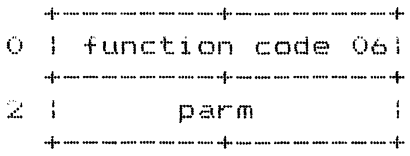
C_PRNOUT sends the character contained in the lower 8 bits of the c parameter to the standard printer device. To ensure compatibility with future extensions to the 16-bit character set, the upper 8 bits of c should be zero.

Function 06: Raw I/O to Standard Console

C Interface:

```
        LONG
c_rawio (parm)
        WORD    parm;
{
}
```

Parameter Block:



Parameters:

parm — Specifies the operation to be performed:

FF To read a character from StdIn
char Character to be sent to StdOut

Return Values:
Register D0.L:

(param = FF)

0 Character not available
(parm = FF)

Description: If parm is FF, C_RAWIO reads a character from the standard input device and returns it it D0.W. If a character is not available from the standard input device D0.L returns zero.

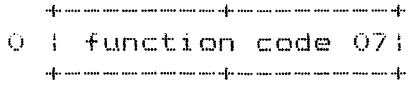
If parm is not FF, C_RAWIO assumes the word contains a character to be sent to the standard output device.

Function 07: Raw Input from Standard Input Device

C Interface:

```
     LONG
c_rawcin ()
{
}
```

Parameter Block:



Return Values:
Register D0.L:

Description: C_RAWCIN reads a character from the standard input device and returns it in D0.L without echoing it to standard output device.

C_RAWCIN does not perform a check for control characters.

Function 08: Read Character from Standard Input, No Echo

C Interface:

```
        LONG
c_necin ()
{
}
```

Parameter Block:

0	function code 08

Return Values:
Register D0.L:

Description: C_NECIN reads a character from the standard input device and returns it in D0.L without echoing it to standard output device.

Control characters are interpreted by this function and have their proper effect; see "Line Editing Commands" in Section 1 of the GEM DOS User's Guide.

Function 09: Write String to Standard Output Device

C Interface:

```
        VOID
c_conws (p)
        BYTE    *p;
{
}
```

Parameter Block:

0	function code 09
2	p
4	

Parameters:

p — Points to a null-terminated string of characters to be written to the standard output device

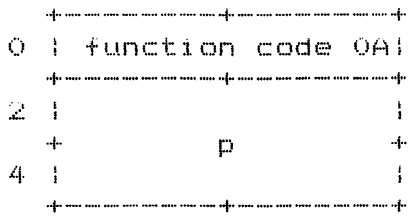
Description: C_CONWS transmits the null-terminated character string pointed to by p to the standard output device, character by character. This function also processes Ctrl-C, which aborts the operation and warm boots the system.

Function 0A: Read Edited String from Standard Input Device

C Interface:

```
        VOID
c_conrs ()
        BYTE    *p;
{
}
```

Parameter Block:



Parameters:

p	Points to an input buffer

Description: C_CONRS reads characters from the caller's default input device and writes them to the input buffer pointed to by the p parameter. The format of the input buffer is shown in Figure 4-3.

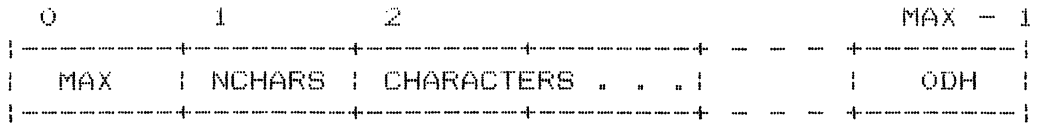


Figure 4-3. C_CONRS Occurrence

Byte 0 of the input buffer specifies the number of characters the buffer can hold. C_CONRS sets byte 1 to the number of characters received. This value does not include the terminator character, a carriage return (0DH). C_CONRS writes the characters it reads from the standard input device into the buffer beginning at the third byte. When the system call reads a carriage return, it stops writing to the buffer.

When the input buffer fills to one less than the value of byte 0, C_CONRS ignores additional input and sounds the bell until it reads a carriage return.

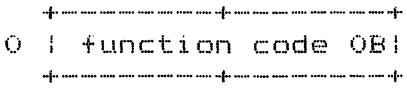
C_CONRS performs line-editing functions on the characters as they are read from the console (Backspace, Ctrl-E, Ctrl-Q, Ctrl-R, Ctrl-S, Ctrl-U, Ctrl-X) and performs a warmboot on Ctrl-C input.

Function 0B: Check Status of Standard Input Device

C Interface:

```
        LONG
c_conis (int)
{
}
```

Parameter Block:



Return Values:
Register D0.L:

0 If no character available

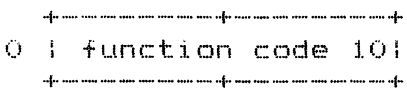
Description: C_CONIS checks the status of the caller's standard input device to determine if a character is ready to be received.

Function 10: Check Status of Standard Output Device

C Interface:

```
        LONG
c_conos ()
{
}
```

Parameter Block:



Return Values:
Register D0.L:

0 If console unavailable

Description: C_CONOS checks the status of the caller's console device to determine if it is ready to receive a character.

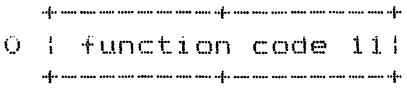
This function also processes Ctrl-C, which aborts the operation and warm boots the system.

Function 11: Check Status of Standard Print Device

C Interface:

```
        LONG
c_prnos ()
{
}
```

Parameter Block:



Return Values:
Register D0.L:

0 If printer unavailable

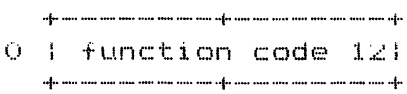
Description: C_PRNOS checks the status of the printer device to determine if it is ready to receive a character.

Function 12: Check Status of Standard Auxiliary Device Input

C Interface:

```
        LONG
c_auxis ()
{
}
```

Parameter Block:



Return Values:
Register D0.L:

0 If no character available

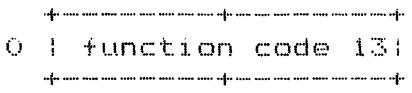
Description C_AUXIS checks the input status of the caller's standard auxiliary device to determine if a character is ready to be received.

Function 13: Check Status of Standard Auxiliary Device Output

C Interface:

```
        LONG
c_auxos ()
{
}
```

Parameter Block:



Return Values:
Register D0.L:

0 If it is unavailable

Description: C_AUXOS checks the status of the caller's auxiliary device to determine if it is ready to receive a character.

4.5 FILE ACCESS FUNCTIONS

Table 4-6 lists the number, mnemonic, and name of the GEM DOS system calls described in this section.

Table 4-6. File Access Function Calls

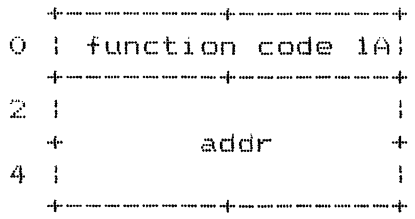
Hex	Mnemonic
1A	F_SETDTA
2F	F_GETDTA
3C	F_CREATE
3D	F_OPEN
3E	F_CLOSE
3F	F_READ
40	F_WRITE
41	F_DELETE
42	F_SEEK
43	F_ATTRIB
45	F_DUP
46	F_FORCE
4E	F_SFIRST
4F	F_SNEXT
56	F_RENAME
57	F_DATIME

Function 1A: Set Disk Transfer Address

C Interface:

```
    VOID
f_setdta (b)
    BYTE *b;
{
}
```

Parameter Block:



Parameters:

addr	Address of DTA buffer

Description: The 44-byte buffer whose location is specified with F_SETDTA may begin on an even or odd address. F_SFIRST (4E) and F_SNEXT (4F) use the DTA to return file search information.

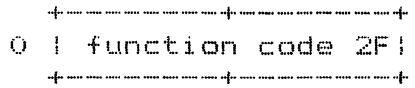
If you do not set the DTA, GEM DOS uses the default DTA at offset 80 in the Base Page. F_GETDTA (2F) returns the current location of the DTA buffer.

Function 2F: Get Disk Transfer Address

C Interface:

```
   LONG
f_getdta ()
{
}
```

Parameter Block:



Return Values:
Register D0.L:

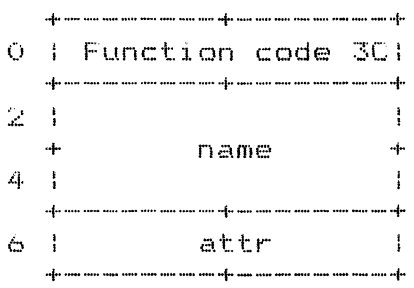
Description: F_GETDTA returns the current address of the DTA buffer in D0.L. You can set the DTA by calling F_SETDTA (1A). Note that only F_SFIRST (4E) and F_SNEXT (4F) use the DTA to return file search information.

Function 3C: Create a File

C Interface:

```
        LONG
f_create (name,attr)
        BYTE        *name;
        WORD        attr;
{
}
```

Parameter Block:



Parameters:

name — Points to a null-terminated string that specifies the drive, path, and name of the file to be created

attr — Specifies the file's attributes:

01	Read-only
02	Hidden
04	System, Hidden
08	Entry is a volume label

Return Values:
Register D0.L:

-34L (Path not found)
-35L (No handles left)
-36L (Access Denied)

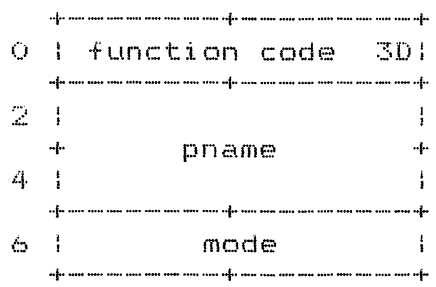
Description: F_CREATE returns error code -36L (Access Denied) to indicate that a file with the name specified in the name string already exists or that the directory is full.

Function 3D: Open a File

C Interface:

```
        LONG
f_open (pname,mode)
        BYTE    *pname;
        WORD mode;
{
}
```

Parameter Block:



Parameters:

pname — Points to a null-terminated string that specifies the drive, path, and name of the file or device to open

mode — File access mode:

0 Read access only
1 Write access only
2 Read/Write access

Return Values:
Register D0.L:

-33L (File not found)
-35L (No handles left)
-36L (Access Denied)

Description: F_OPEN positions the file pointer at the beginning of the file. You can change the pointer's location with the F_SEEK (42) function call.

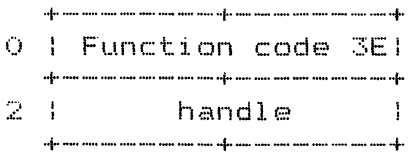
Use the handle returned by F_OPEN to refer to the file for subsequent read, write, seek, and close operations. You can use F_ATTRIB (43) to get the file's attribute and F_DATIME to get or set its date and time stamps.

Function 3E: Close a File

C Interface:

```
        LONG
f_close (handle)
        WORD      handle;
{
}
```

Parameter Block:



Parameters:

handle	File handle of file to be closed

Return Values:
Register D0.L:

-37L (Invalid handle)

Description: The handle you specify in the parameter block on entry to F_CLOSE should be one that was previously returned by F_OPEN (3D).

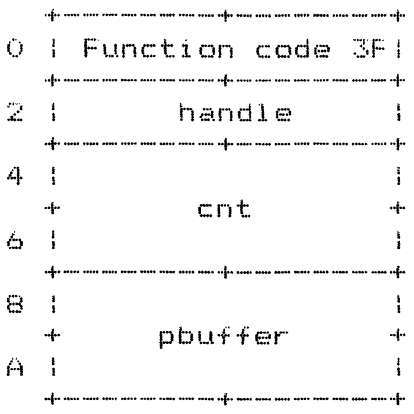
All disk buffers associated with the file are flushed and the file's directory entry is updated on successful return from this call.

Function 3F: Read from a File

C Interface:

```
        LONG
f_read (handle, cnt, pbuffer)
     WORD       handle;
     LONG       cnt;
     BYTE       *pbuffer;
{
}
```

Parameter Block:



Parameters:

handle	File handle (returned from F_OPEN) of file to be read
cnt	Number of bytes to be read
pbuffer	Points to the buffer into which the file's contents are to be read

Return Values:
Register D0.L:

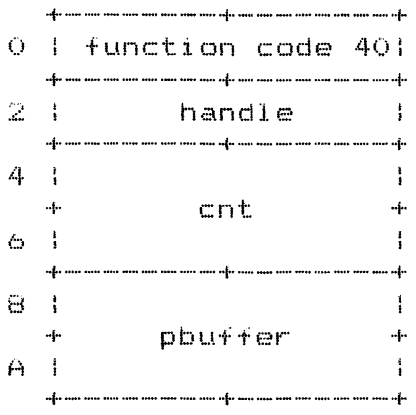
-37L (Invalid handle)

Function 40: Write to a File

C Interface:

```
        LONG
f_write (handle, cnt, pbuffer)
        WORD        handle;
        LONG        cnt;
        BYTE        *pbuffer;
{
}
```

Parameter Block:



Parameters:

handle	File handle (returned from F_OPEN)
cnt	Number of bytes to be written
pbuffer	Points to the buffer that contains the bytes to be written

Return Values:
Register D0.L:

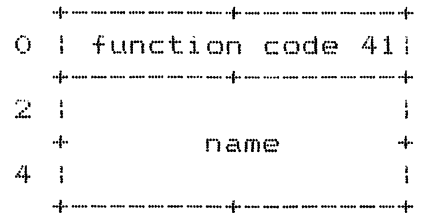
-36L (Access Denied)
-37L (Invalid handle)

Function 41: Delete a File

C Interface:

```
        LONG
f_delete (name)
        BYTE        *name;
{
}
```

Parameter Block:



Parameters:

name — Points to a null-terminated string that specifies the drive, path, and name of the directory entry to be deleted

Return Values:
Register D0.L:

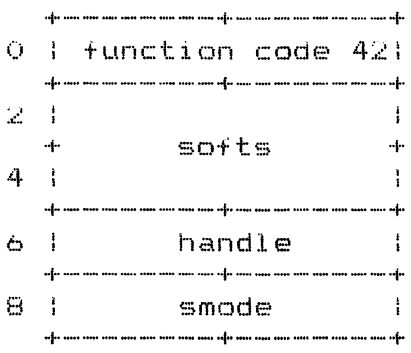
-33L (File not found)
-36L (Access Denied)

Function 42: Seek File Pointer

C Interface:

```
        LONG
f_seek (softs, handle, smode)
        LONG    softs;
        WORD    handle;
        WORD    smode;
{
}
```

Parameter Block:



Parameters:

softs A signed value, N, used to indicate byte offset negative values are useful in modes 1 and 2 below +N moves pointer toward end of file, -N toward beginning of file

handle File handle (returned from F_OPEN)

smode Indicates how the file pointer is to be moved:

0 Move pointer N bytes from the beginning of the file
1 Move pointer N bytes from its current location
2 Move pointer N bytes from the end of the file

Return Values:
Register D0.L:

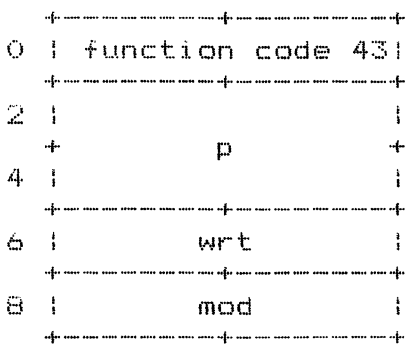
from beginning of file)
-37L (Invalid handle)
-32L (Invalid function)

Function 43: Get/Set File Attributes

C Interface:

```
        LONG
f_attrib (p, wrt, mod)
        BYTE     *p;
        WORD     wrt,mod;
{
}
```

Parameter Block:



Parameters:

p — Points to a null-terminated string that contains the drive, path, and file name

wrt — Specifies the operation to be performed:
0 Return the file's attributes in D0.L
1 Set the file's attributes according to mod

mod — Indicates file attributes:
01H Read-only
02H Hidden from directory search
04H System, hidden from directory search
08H Entry contains the volume label
10H Entry is a subdirectory
20H Archive (file written to and closed)

Return Values:
Register D0.L:

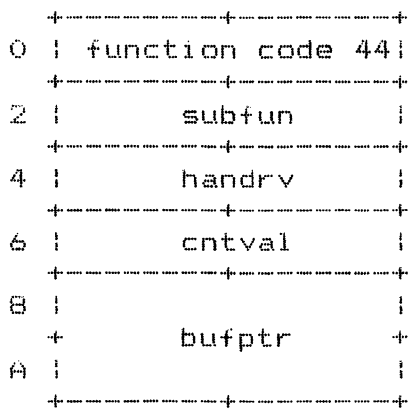
-33L (File not found)
-34L (Path not found)

Function 44: I/O Control

C Interface:

```
    LONG
f_ioctl (subfun, hndrv, cntval, bufptr)
    WORD subfun, hndrv, cntval;
    LONG bufptr;
{
}
```

Parameter Block:



Parameters:

subfun — F_IOCTL subfunction code value as follows:

-1	Substitute interrupt vector
0	Get device information
1	Reserved
2	Read from device control channel
3	Write to device control channel
4	Read from drive control channel
5	Write to drive control channel
6	Get input status
7	Get output status
8	Get media type

hndrv — Handle or drive code, subfunction dependent. See Table 4-7, below.

cntval — Byte count or bit map of values, subfunction dependent.

bufptr — Pointer to buffer or, for subfunction -1, the address of device's interrupt vector.

Return Values:

Device's current interrupt vector for subfunction -1
Bit map of device information for subfuction 0
Number of bytes transferred for subfunctions 2-5
Disk file or character device input status for subfunction 6
Disk file or character device output status for subfunction 7
Media type for subfunction 8

Description: On entry to F_IOCTL, offset 02 of the parameter block contains a subfunction code value that determines the operation to be performed. Other entry parameters vary according to the subfunction you call. Table 4-7 lists the F_IOCTL subfunction calling restrictions. Descriptions of the individual subfunctions follow the table.

Table 4-7. F_IOCTL Calling Restrictions

Subfunction Number	Character Device Handle	Standard Handle	Standard Handle	Disk File Number	Drive	Byte Count/Buffer
-1	X					
0	X	X	X			
2	X	X			X	
3	X	X			X	
4				X	X	
5				X	X	
6	X	X	X			
7	X	X	X			
8				X		

Subfunction -1: Specify a character device handle at offset 04, 0 at offset 06, and the address of the device's interrupt handler at offset 08 of the F_IOCTL parameter block. Subfunction -1 returns the current interrupt vector for the device whose handle you specify. Note that only a character device handle can be specified for this subfunction.

Subfunction 0: Specify a disk file or character device handle at offset 04 of the F_IOCTL parameter block. Subfunction 0 returns the bit map of device information shown below or an error number (-35L, -37L) in register D0.L.

If bit 7 = 0:

Bits	Meaning
0-5	Drive number
6	Reserved
7	0 = Disk file
8-D	Reserved
E	1 = Device can process control strings
F	Reserved

If bit 7 = 1:

Bits	Meaning
0	1 = CON: input (stdin)
1	1 = CON: output (stdout)
2	1 = NUL:
3	1 = CLK:
4-6	Reserved
7	1 = Character device
8-D	Reserved
E	1 = Device can process control strings
F	Reserved

Subfunction 2: Specify the handle of a character device at offset 4, the number of bytes to be read at offset 6, and the address of the buffer into which this subfunction is to read characters from the device's control channel at offset 8 of the F_IOCTL parameter block.

Subfunction 2 returns the number of bytes read or an error number in register D0.L.

Subfunction 3 Specify the handle of a character device at offset 4, of the number of bytes to be written to the device's control channel at offset 6, and the address of the buffer from which the bytes are to written at offset 8 of the F_IOCTL parameter block.

Subfunction 3 returns the number of bytes written to the device control channel or an error number in register D0.L.

Subfunction 4 Specify the number of the drive at offset 4, the number of bytes to be read from the drive's control channel at offset 6, and the address of a buffer at offset 8 of the F_IOCTL parameter block.

The drive number can be 0 for the default drive, 1 for drive A, and so on up to the number of drives supported by your particular system.

Subfunction 4 returns the number of bytes read from the drive's control channel or an error number in register D0.L

Subfunction 5 Specify the number of the drive at offset 4, the number of bytes to written to the drive's control channel at offset 6, and the buffer from which the bytes are to be written at offset 8 of the F_IOCTL parameter block.

Subfunction 5 returns the number of bytes written to the drive's control channel or an error number in register D0.L.

Subfunction 6 Specify the handle of a disk file or device whose input status you want returned at offset 4 of the F_IOCTL parameter block. If the handle you specify is a disk file, subfunction 6 returns zero in D0.L when the file pointer is at EOF, or a non-zero value. If the file pointer is repositioned after reaching EOF, this subfunction returns a non-zero value again.

If the handle you specify is for a device, subfunction 6 returns a non-zero value in D0.L to indicate a ready state, or zero to indicate that the device is not ready.

Subfunction 7 Specify the handle of a disk file or device whose output status you want returned at offset 4 of the F_IOCTL parameter block. If the handle you specify is for a disk file, subfunction 7 returns zero in D0.L when the file pointer is at EOF, or a non-zero value. If the pointer is repositioned after reaching EOF, this subfunction returns non-zero again.

If the handle you specify is for a device, subfunction 7 returns a non-zero value to indicate a ready state, or a zero value to indicate that the device is not ready for output.

Subfunction 8 Specify the number of the drive whose media type you returned at offset 4 of the F_IOCTL parameter block. If the drive number you specify is valid, this subfunction returns zero to indicate that the drive contains removable media or a non-zero value to indicate that the drive contains fixed media.

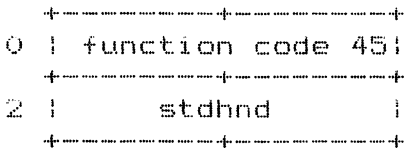
The drive number you specify may be 0 for the default drive, 1 for drive A, and so on up to the number of drives supported by your particualr system.

Function 45: Duplicate a File Handle

C Interface:

```
    LONG
f_dup (stdhnd)
    WORD stdhnd;
{
}
```

Parameter Block:



Parameters:

stdhnd Standard file handle

Return Values:
Register D0.L:

-35L (No handles left)
-37L (Invalid handle)

Description: F_DUP accepts a standard file handle as input and returns a non-standard handle that refers to the same file or device at the same position.

If you use F_SEEK (42) to change the location of the pointer for either handle, the pointer of its duplicate is also moved.

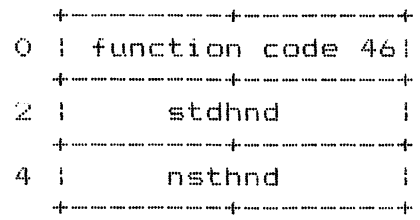
See Table 4-2 for the GEM DOS standard handle values.

Function 46: Force File Handle

C Interface:

```
    LONG
f_force (stdhnd, nsthnd)
    WORD stdhnd, nsthnd;
{
}
```

Parameter Block:



Parameters:

stdhnd — Standard file handle

nsthnd — Non-standard file handle

Return Values:
Register D0.L:

-37L (Invalid handle)

Description: F_FORCE forces the standard device handle specified at offset 02 of the parameter block to point to the same file or device as the non-standard handle specified at offset 04.

If the file whose handle you specify in stdhnd is open, F_FORCE closes before returning. If you use F_SEEK (42) to change the location of the pointer for either handle, the pointer of its duplicate is also moved.

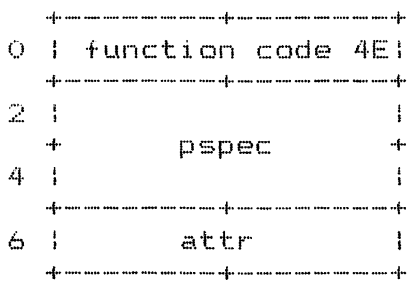
See Table 4-2 for the GEM DOS standard handle values.

Function 4E: Search for First Occurrence of Filespec

C Interface:

```
     LONG
f_sfirst (pspec, attr)
     BYTE      *pspec;
     WORD      attr;
{
}
```

Parameter Block:



Parameters:

pspec — Points to a null-terminated string that specifies the drive, path and name of the file to search for; only the file name portion of pspec may contain * or ? wildcards

attr — Specifies the search attributes:

00H File is normal file entry
01H File is read-only
02H File is hidden from directory search
04H File is system file
08H Entry is a volume label
10H Entry is a subdirectory
20H File's archive bit is set

Return Values:
Register D0.L:

-33L (File not found)
-49L (No more files)

Description: F_SFIRST searches for the first directory entry that matches pspec according to the value of attr.

F_SFIRST uses the value of attr in the following manner:

* If attr is 00H, the search is restricted to normal directory entries. F_SFIRST does not return entries for volume labels, subdirectories, hidden files, or system files.

* If attr is set to the hidden (02H) or system (04H) file values, they are included with the normal entries in the search set. You can use F_SFIRST to search for all directory entries except volume labels by setting attr for hidden, system, and directory.

* If attr is set to the volume label value (08H), F_SFIRST restricts its search to volume labels.

If F_SFIRST finds a directory entry that matches pspec and attr, it formats the DTA as shown in Table 4-8. F_GETDTA (2F) returns the current location of the DTA buffer.

Table 4-8. DTA Contents from F_SFIRST Call

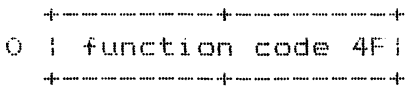
DTA Offset	Contents
0 - 20	Reserved for OS use
21	File's attributes
22 - 23	File's time stamp
24 - 25	File's date stamp
26 - 29	Longword of file's size
30 - 43	File's name and extension

Function 4F: Search for Next Occurrence of Filespec

C Interface:

```
        LONG
f_snext()
{
}
```

Parameter Block:



Return Values:
Register D0.L:

-49L (No more files)

Description

F_SNEXT uses the parameters specified in a previous F_SFIRST (4E) or F_SNEXT call to locate the next matching directory entry. Offsets 0 to 20 of the DTA must remain untouched from the previous SFIRST or SNEXT call.

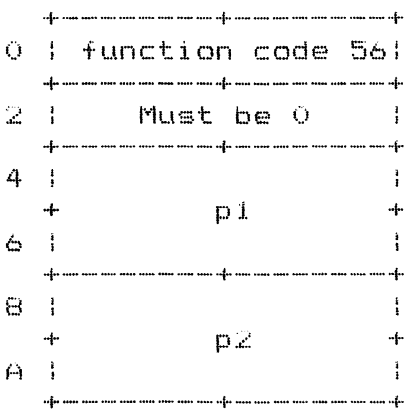
If F_NEXT locates a matching entry, it updates the contents of the DTA buffer initialized by the previous F_SFIRST or F_NEXT call as shown in Table 4-8, above.

Function 56: Rename a File

C Interface:

```
        LONG
f_rename (res, p1, p2)
        WORD    res;
        BYTE    *p1;
        BYTE    *p2;
{
}
```

Parameter Block:



Parameters:

p1 — Points to a null-terminated string that contains the drive, path, and name of the file to be renamed

p2 — Points to a null-terminated string that contains the path and new name for the file specified in p1

Return Values:
Register D0.L:

-34L (Path not found)
-36L (Access denied)

Description: The word at offset 2 in the parameter block is reserved for system use and must be zero. The file name specified in the p2 string must not exist. If you specify a drive in the p2 string, it must be the same drive specified in the string pointed to by p1.

You can specify a different directory path in the p2 string so that the operation moves the file to another subdirectory on the same drive.

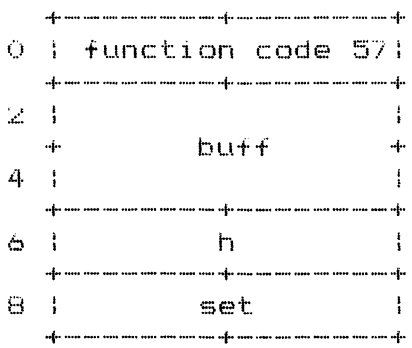
F_RENAME returns -36L in D0 if the read/only attribute of the file to be renamed is set.

Function 57: Get/Set File Date and Time Stamps

C Interface:

```
     VOID
f_datime (buf, h, set)
     BYTE      *buff;
     WORD      h;
     WORD      set;
{
}
```

Parameter Block:



Parameters:

buff	Points to a buffer that contains the time in the first word and the date in the second word
h	Contains the file handle returned from F_OPEN
set	A flag that specifies the operation to be performed:

0 Set date and time
1 Return date and time to buff

Description: The format of the date and time contained in the buffer pointed to by buff is as follows:

Time (word 0):
Bits 00 - 04
Bits 05 - 10
Bits 11 - 15

Date (word 1):
Bits 00 - 04
Bits 05 - 08
Bits 09 - 15

4.6 DRIVE AND DIRECTORY MANAGEMENT FUNCTIONS

Table 4-9 lists the number, mnemonic, and name of the GEMDOS drive and directory management functions.

Table 4-9. Drive and Directory Management Functions

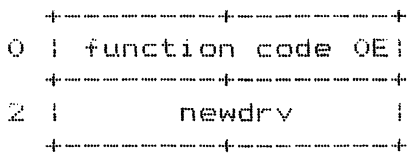
Hex	Mnemonic
0E	D_SETDRV
19	D_GETDRV
36	D_FREE
39	D_CREATE
3A	D_DELETE
3B	D_SETPATH
47	D_GETPATH

Funtion 0E: Set Default Drive

C Interface:

```
LONG
d_setdrv (newdrv)
WORD newdrv;
{
}
```

Parameter Block:



Parameters:

newdrv Specifies the number for the current drive:

```
0     Drive A
1     Drive B
3     Drive C
.
.
.
15    Drive P
```

Return Values:
Register D0.L:

```
Bit  0 = A
Bit  1 = B
Bit  2 = C
.       .
.       .
.       .
Bit 15 = p
```

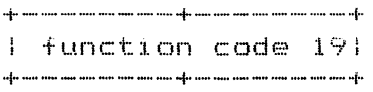
Description: D_SETDRV designates the drive whose number is specified at offset 04 in the parameter block as the current drive and returns a bit map of the system drives in D0.L.

Function 19: Get Default Drive

C Interface:

```
    LONG
d_getdrv ()
{
}
```

Parameter Block:



Return Values:
Register D0.L:

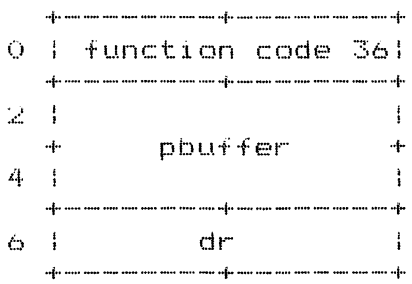
Description: D_GETDRV returns the number of the current drive in D0.L, where: 0 = A, 1 = B, 2 = C, up to 15 = P

Function 36: Get Drive Free Space

C Interface:

```
    LONG
d_free (pbuffer,dr)
    BYTE *pbuffer
    WORD dr;
{
}
```

Parameter Block:



Parameters:

pbuffer — Points to the buffer into which D_FREE returns drive information

dr — Specifies the drive code where:

0 = Current Drive, 1 = Drive A, 2=Drive B, etc.

Return Values:
Register D0.L:

-46L (Invalid drive specified)

Description: The format of the buffer pointed to by pbuffer on successful return from D_FREE is shown in Figure 4-4.

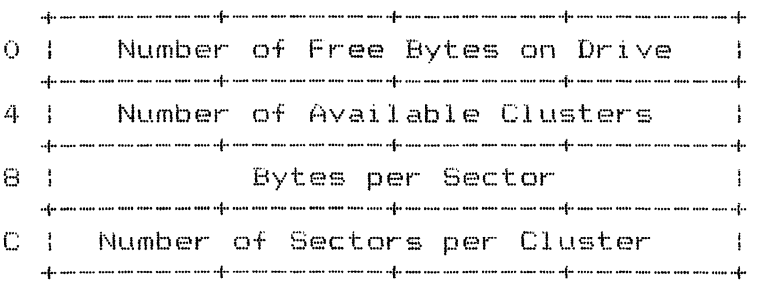


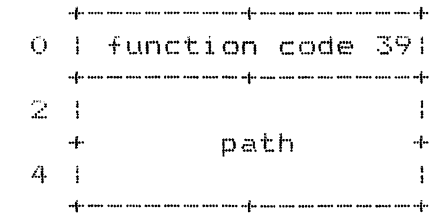
Figure 4-4. D_FREE Information Return Buffer

Function 39: Create a Subdirectory

C Interface:

```
        LONG
d_create (path),
        BYTE    *path;
{
}
```

Parameter Block:



Parameters:

path — Points to a null-terminated string that contains the drive and pathname of the subdirectory to be created

Return Values:
Register D0.L:

-34L (Path not found)
-36L (Access denied)

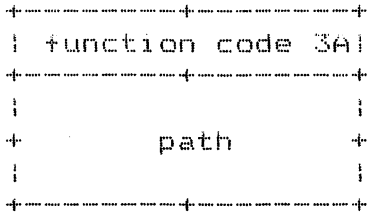
Description: D_CREATE does not change the directory path if any element of the pathname contained in the path string does not exist. If D0.L contains 0 on return from D_CREATE, the new subdirectory is created at the end of the pathname.

Function 3A: Delete a Subdirectory

C Interface:

```
        LONG
d_delete (path)
        BYTE    *path;
{
}
```

Parameter Block:



Parameters:

path — Points to a null-terminated string that contains the drive and pathname of the subdirectory to be deleted

Return Values:
Register D0.L:

-34L (Path not found)
-36L (Access denied)
-65L (Internal error)

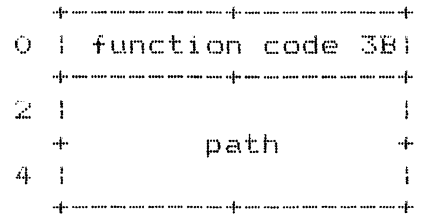
Description: If the subdirectory to be deleted is not empty, D_DELETE returns -36L (Access denied). D_DELETE also returns an error code if you attempt to delete the current or root directories.

Function 3B: Set Current Directory

C Interface:

```
        LONG
d_setpath (path)
        BYTE        *path;
{
}
```

Parameter Block:



Parameters:

path	Points to a null-terminated string that contains the drive and pathname of the directory to be made current

Return Values:
Register D0.L:

-34L (Path not found)

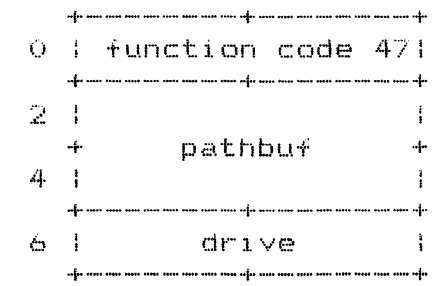
Description: If D0.L contains 0 on return from D_SETPATH, the current directory is set to correspond with the path string.

Function 47: Get Current Directory

C Interface:

```
        LONG
d_getpath (pathbuf, drive)
        BYTE    *pathbuf;
        WORD    drive;
{
}
```

Parameter Block:



Parameters:

pathbuf — Points to a 64-byte buffer into which D_GETPATH writes the complete pathname of the current directory

drive — Contains the drive code where:
0 = current drive, 1 = drive A, 2 = drive B, etc.

Return Values:
Register D0.L:

-46L (Invalid drive)

Description: D_GETPATH writes the pathname (beginning from the root) of the current directory for the drive specified in the parameter block into the null-terminated buffer pointed to by pathbuf. D_GETPATH does not include the drive letter in the pathname nor does it begin the pathname with a backslash (\).

4.7 PROGRAM CONTROL FUNCTIONS

Table 4-10 lists the number, mnemonic, and name of the GEM DOS system calls described in this section.

Table 4-10. System and Program Control Function Calls

Hex	Mnemonic
00	P_TERM0
25	S_SETVEC
30	S_VERSION
31	P_TERMRES
35	S_GETVEC
4B	P_EXEC
4C	P_TERM

Function 00: Program Terminate

C Interface:

```
VOID
p_term0 ()
{
}
```

Parameter Block:

0	function code 00

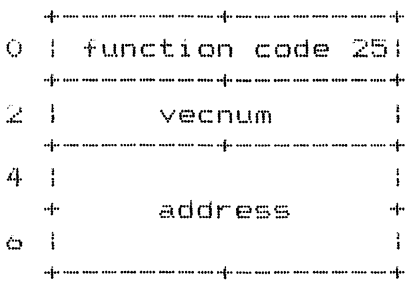
Description: P_TERM0 terminates the calling process and returns to the parent process with a return code of 0.

Function 25: Set Exception Vector

C Interface:

```
        LONG
s_setvec (vecnum, address)
        WORD vecnum;
        LONG address;
{
}
```

Parameter Block:



Parameters:

vecnum	Vector number
address	New address for vector number

Return Values:
Register D0.L:

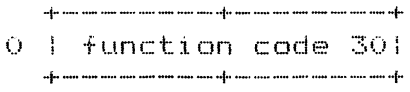
Description: S_SETVEC associates a vector number with the specified address.

Function 30: Get Version Number

C Interface:

```
    WORD
s_version ()
{
}
```

Parameter Block:



Return Values:
Register D0.W:

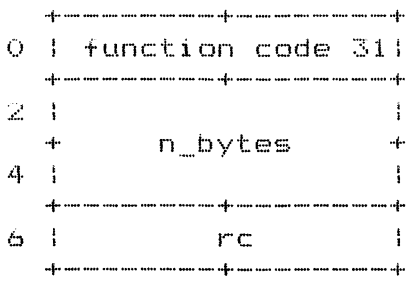
Description: The lower byte of the word returned in D0 contains the major version number; the upper byte contains the minor version number. 0001 = version 1.00.

Function 31: Terminate and Stay Resident

C Interface:

```
    VOID
p_termres (n_bytes,rc)
    LONG n_bytes
    WORD rc;
{
}
```

Parameter Block:



Parameters:

n_bytes	Memory size in bytes
rc	Exit code

Description: P_TERMRES allows the calling process to terminate the currently running process and leave the amount of memory specified in n_bytes (offset 02) resident.

The rc field at offset 06 of the P_TERMRES parameter block contains an exit code that is returned to the parent process in register D0.

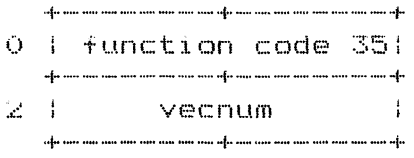
Note that your use of P_TERMRES might make your program difficult to port to future versions of this operating system.

Function 35: Get Exception Vector

C Interface:

```
        LONG
s_setvec (vecnum)
        WORD vecnum;
{
}
```

Parameter Block:



Parameters:

vecnum	Vector number

Return Values:
Register D0.L:

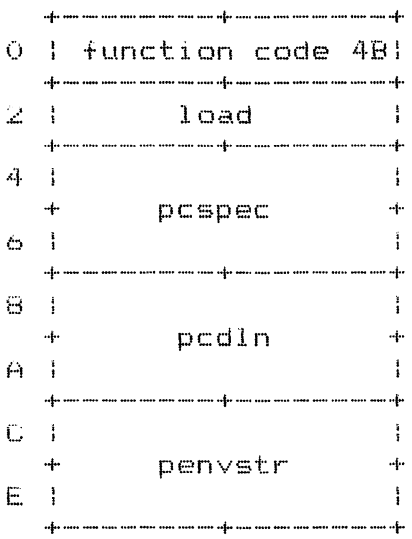
Description: S_GETVEC returns the address associated with the specified vector number.

Function 4B: Load or Execute a Process

C Interface:

```
        LONG
p_exec (load, pcspec, pcmdln, penvstr)
        WORD        load;
        BYTE        *pcspec;
        BYTE        *pcdln;
        BYTE        *penvstr;
{
}
```

Parameter Block:



Parameters:

load — Indicates the type of load function to be performed:

- 0 Load and execute program
- 3 Load overlay, do not execute

pcspec — Points to a null-terminated string that specifies the drive, path, and name of the file to be loaded

pcdln — Points to a command tail that includes redirection information

penvstr — Points to the ASCII strings that describe the configuration of the process's environment

Return Values:
Register D0.L:

on load/execute (load = 0)

Base page address
on load overlay (load=3)

-33L (File not found)
-39L (Insufficient memory)
-66L (Invalid program load format)

Description: The address contained in the pcdln field is placed at offset 80H in the base page of the program being loaded.

The process environment strings pointed to by the penvstr field at offset 0C in the P_EXEC parameter block are formatted as a series of null-terminated strings. These environment strings are terminated by a null word. The address contained in the penvstr field is copied into the p_env field at offset 2C of the base page.

The child process exit code returned by P_TERMRES in D0.L can be specified at offset 02 in the parameter block used to invoke Terminate Process (4CH).

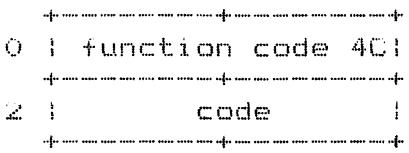
See Section for a description of the base page format.

Function 4C: Terminate Process

C Interface:

```
      VOID
p_term (code)
WORD code;
{
}
```

Parameter Block:



Parameters:

code — Exit code returned to parent process from Load or Execute a Process (4B).

Return Values:
Register D0.L:

Error code on failure

Description: P_TERM terminates the current process and transfers control to the calling process. All files opened by the terminating process are closed.

4.8 MEMORY MANAGEMENT FUNCTIONS

This section describes the following GEM DOS function calls:

* Allocate Memory (M_ALLOC, 48)
* Free Allocated Memory (M_FREE, 49)
* Shrink Size of Allocated Memory (M_SHRINK, 4A)

M_FREE (49) and M_SHRINK calls require the address of a memory block previously returned by M_ALLOC (48) as a parameter. M_ALLOC allocates memory in units of bytes and can also be used to return the amount of currently available memory.

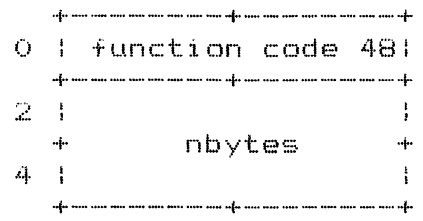
The GEM DOS command line interpreter allocates all of available memory to most of the programs that are loaded under its control.

Function 48: Allocate Memory

C Interface:

```
        LONG
m_alloc (nbytes)
        LONG    nbytes;
{
}
```

Parameter Block:



Parameters:

nbytes — The number of bytes to allocate or -1L (FFFFFFFF) to return the maximum amount of available memory

Return Values:
Register D0.L:

block's starting address

Number of available bytes
(nbytes = FFFFFFFF)

0 on failure

Description: If the nbytes field of the M_ALLOC parameter block specifies the number of bytes of memory to allocate, the function call returns a pointer to the starting address of a memory block in D0.L. If the allocation operation fails, D0.L returns zero.

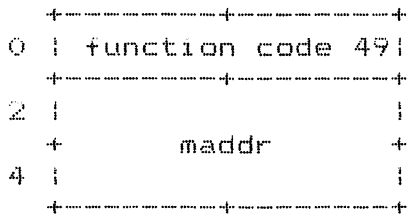
If you set nbytes to -1L (FFFFFFFF), M_ALLOC returns the maximum number of available bytes.

Function 49: Free Allocated Memory

C Interface:

```
        LONG
m_free (maddr)
        LONG    maddr;
{
}
```

Parameter Block:



Parameters:

maddr Starting address of memory block to be freed

Return Values:
Register D0.L:

-40L (Invalid memory block address)

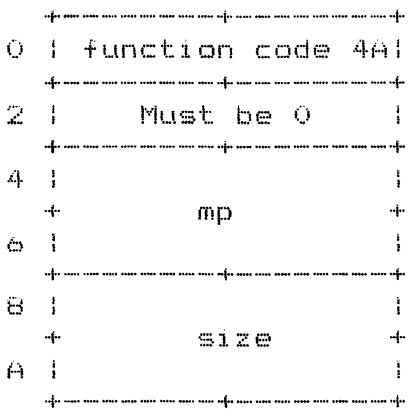
Description: M_FREE returns a memory block previously allocated by M_ALLOC (48) to the system. If the maddr field of the parameter block does not specify an address that was returned by a previous M_ALLOC call, M_FREE returns -40L in D0.L to indicate that you have used an invalid memory block address.

Function 4A: Shrink Size of Allocated Memory

C Interface:

```
    LONG
m_shrink (res, mp, size)
    res;
    BYTE *mp;
    LONG size;
{
}
```

Parameter Block:



Parameters:

mp	Beginning address of memory block to be returned
size	Length, in bytes, of memory to be returned

Return Values:

-40L (Invalid memory block address)
-67L (Growth restriction failure)

Description: The word at offset 2 in the parameter block is reserved for system use and must be zero. mp must specify a memory block address that was previously returned by M_ALLOC (48). If the value you pass to M_SHRINK in the size field of the parameter block does not specify a reduction in the memory block size, M_SHRINK returns -67L in D0.L

Use M_SHRINK to return memory that your program does not require to the system for allocation to other processes. Before calling M_SHRINK, move your program's stack space down into an area of memory that you have previously reserved and know will accomodate your program's stack size.

If your program later requires additional memory, call M_ALLOC to determine the amount of memory available (nbytes = -1). Then call M_ALLOC with nbytes set to the portion of available memory your requires.

4.9 TIME FUNCTIONS

This section describes the Get Date (2A), Set Date (2B), Get Time (2C), and Set Time (2D) functions.

Function 2A: Get Date

C Interface:

```
WORD
t_getdate ()
{
}
```

Parameter Block:

0	function code 2A

Return Values:
Register D0.W:

Bits 00 - 04 Day, 1 - 31
Bits 05 - 08 Month, 1 - 12
Bits 09 - 15 Year (since 1980), 0 - 119

Function 2B: Set Date

C Interface:

```
   LONG
t_setdate (date)
   WORD date;
{
}
```

Parameter Block:

0	function code 2B
2	date

Parameters:

date Contains the date to be set in the format:

Bits 00 - 04 Day, 1 - 31
Bits 05 - 08 Month, 1 - 12
Bits 09 - 15 Year (since 1980), 0 - 119

Return Values:
Register D0.L:

-1L (Error) if date is not valid

Function 2C: Get Time

C Interface:

```
   WORD
t_gettime ()
{
}
```

Parameter Block:

0	function code 2C

Return Values:
Register D0.W:

Bits 00 - 04 Binary number of two-second increments
Bits 05 - 10 Binary number of minutes
Bits 11 - 15 Binary number of hours

Function 2D: Set Time

C Interface:

```
   LONG
t_settime (time)
   WORD time;
{
}
```

Parameter Block:

0	function code 2D
2	time

Parameters:

time Contains the time to be set in the format:

Bits 00 - 04 Binary number of two-second increments
Bits 05 - 10 Binary number of minutes
Bits 11 - 15 Binary number of hours

Return Values:
Register D0.L:

-1L (Error) if time is not valid

AS68 ASSEMBLER

5.1 ASSEMBLER OPERATION

The GEM DOS Assembler, AS68, assembles an assembly language source program for execution on the 68000 microprocessor. It produces a relocatable object file and, optionally, a listing. You can find a summary of the AS68 instruction set at the end of this section. Exceptions and additions to the standard Motorola instruction set appear in sections 5.6 and 5.7.

5.2 INITIALIZING AS68

If the file AS68SYM.DAT is not on your disk, you must create this file to initialize AS68 before you can use AS68 to assemble files. To initialize AS68, specify the AS68 command, the -I option, and the filename AS68INIT as shown below.

```
(a)AS68 -I AS68INIT
```

AS68 creates the output file AS68SYMB.DAT, which AS68 requires when it assembles programs. After you create this file, you need not specify this command line again unless you reconfigure your system to have different TPA boundaries.

5.3 INVOKING THE ASSEMBLER (AS68)

Invoke AS68 by entering a command with the following form:

```
AS68 [-F pathname] [-P] [-S pathname] [-U] [-L] [-N] [-I]
[-O object filename]
source pathname [>listing pathname]
```

Table 5-1 lists and describes the AS68 command line options.

Table 5-1. Assembler Options

Option	Meaning
-F pathname	Specifies the directory in which the temporary files are created. If this option is not specified, AS68 creates the temporary files in the current default directory.
-I	Initializes the assembler. See Section 5.2 for details.
-P	If specificed, AS68 produces and prints a listing on the standard output device, which, by default, is the console. Redirect the listing, including error messages, to a file with the >listing filename parameter. Note that error messages are produced whether or not the -P option is specified. No listing is produced, however, unless you specify the -P option.
-S pathname	Indicates the directory that contains the assembler initialization file, AS68SYMB.DAT. This file is created when you initialize AS68. AS68 reads the file AS68SYMB.DAT before it assembles a source file. If you do not specify this option, AS68 assumes the initialization file is located in the current default directory.
-U	Causes all undefined symbols in the assembly to be treated as global references.
-L	Ensures all address constants are generated as longwords. Use the -L option for programs that require more than 64K for execution or if the TPA is not contained in the first 64K bytes of memory. If -L is not specified, the program is assembled to run in the first 64K bytes of memory. If an address in the assembly does not fit within one word, an error occurs.
-N	Disables optimization of branches on forward references. Normally, AS68 uses the 2-byte form of the conditional branch and the 4-byte BSR instruction wherever possible (instead of the 6-byte JSR instruction) to speed program execution and reduce instruction size.
-T	Enables AS68 to accept the 68010 microprocessor opcodes.
source filename	Specifies the file to assemble; you must supply this parameter.
>listing filename	Sends a program listing to the standard output device. Use the greater-than symbol, >, to direct the listing to a disk file. The listing includes assembler error messages. Note that if you do not specify -P with a listing filename, only the error messages are redirected to the listing file.

5.4 ASSEMBLY LANGUAGE DIRECTIVES

Table 5-2 lists the AS68 directives.

Table 5-2. AS68 Directives

Directive — Meaning

.comm label, expression
: The comm (common) directive specifies a label and the size of a common area that programs assembled separately can share. The LO68 linker links all areas with the same label to the same address. The largest common area of a group with the same label determines the final size of the program common area.

.data
: The data directive instructs AS68 to change the assembler base segment to the data segment.

.bss
: The bss (block storage segment) directive instructs AS68 to change the assembler base segment to the block storage segment. You cannot assemble instructions and data in the bss. However, you can define symbols and reserve storage in the bss with the ds directive.

.dc operand[,operand,...]
: The dc (define constant) directive defines one or more constants in memory. The operands can be symbols or expressions assigned numeric values by AS68, or explicit numeric constants in decimal or hexadecimal, or strings of ASCII characters. You must separate operands with commas. You must enclose string constants in single quotation marks. Each ASCII character is assigned a full byte of memory. The eighth bit is always 0.

  You can specify the length of each constant with a single letter parameter (byte = b, word = w, longword = l). You must separate the letter from the dc with a period as shown in the following explanations.

  .dc.b
  : The constants are byte constants. If you specify an odd number of bytes, AS68 fills the odd byte on the right with zeros unless the next statement is another dc.b directive. When the next statement is a dc.b directive, the dc.b uses the odd byte. Byte constants are not relocatable.

Table 5-2. (continued)

Directive	Meaning
.dc.w	The constants are word constants. If you specify an odd number of bytes, AS68 fills the last word on the right with zeros to force an even byte count. The only way to specify an odd number of bytes is with an ASCII constant. Word constants can be relocated.
.dc.l	The constants are longword constants. If less than a multiple of four bytes is entered, AS68 fills the last longword on the right with zeros to force a multiple of four bytes. Longword constants can be relocated.
.ds operand	The define storage directive (ds) reserves memory locations. The contents of the memory that it reserves is not initialized. The operand specifies the number of bytes, words, or longwords that this directive reserves. The notation for these size specifications is shown below.
.ds.b	reserves memory locations in bytes
.ds.w	reserves memory locations in words
.ds.l	reserves memory locations in longwords
.end	The end directive informs AS68 that no more source code follows this directive. Code, comments, or multiple carriage returns cannot follow this directive.
.endc	The endc directive denotes the end of the code that is conditionally assembled. It is used with other directives that conditionally assemble code.
.equ (or =) expression	The equate directive (equ or =) assigns the value of the expression in the operand field to the symbol in the label field that precedes the directive. The syntaxes for the equate directive are label .equ expression label = expression

Table 5-2. (continued)

Directive	Meaning
	The label and operand fields are required. The label must be unique; it cannot be defined anywhere else in the program. The expression cannot include an undefined symbol or one that is defined following the expression. Forward references to symbols are not allowed for this directive.
.even	The even directive increments the location counter to force an even boundary. For example, if specified when the location counter is odd, the location counter is incremented by one so that the next instruction or data field begins on an even boundary in memory.
.globl label[,label...] .xdef label[,label...] .xref label[,label...]	These directives make the label(s) external. If the labels are defined in the current assembly, this statement makes them available to other routines during a load by LO68. If the labels are not defined in the current assembly, they become undefined external references, which LO68 links to external values with the same label in other routines. If you specify the -U option, the assembler makes all undefined labels external.
.ifeq expression .ifne expression .ifle expression .iflt expression .ifge expression .ifgt expression	These directives test an expression against zero for a specified condition. If the expression is true, the code following is assembled; if false, the code is ignored until an end conditional directive (endc) is found. The directives and the conditions they test are: .ifeq — equal to zero .ifne — not equal to zero .ifle — less than or equal to zero .iflt — less than zero .ifge — greater or equal to zero .ifgt — greater than zero

Table 5-2. (continued)

Directive	Meaning
.ifc 'string1', 'string2' .ifnc 'string1', 'string2'	The conditional string directive compares two strings. The 'c' condition is true if the strings are exactly the same. The 'nc' condition is true if they do not match.
.offset expression	The offset directive creates a dummy storage section by defining a table of offsets with the define storage directive (ds). The storage definitions are not passed to the linker. The offset table begins at the address specified in the expression. Symbols defined in the offset table are internally maintained. No instructions or code-generating directives, except the equate (equ) and register mask (reg) directives, can be used in the table. The offset directive is terminated by one of the following directives: bss data end section text
.org expression	The absolute origin directive (org) sets the location counter to the value of the expression. Subsequent statements are assigned absolute memory locations with the new value of the location counter. The expression cannot contain any forward, undefined, or external references.
.page	The page directive causes a page break which forces text to print on the top of the next page. It does not require an operand or a label and it does not generate machine code. The page directive allows you to set the page length for a listing of code. If you use this directive and print the source code by specifying the -P option in the AS68 command line, pages break at predefined rather than random places. The page directive does not appear on the printed program listing.

Table 5-2. (continued)

Directive	Meaning
.reg reglist	The register mask directive builds a register mask that can be used by a movem instruction. (See Table 1-1.) One or more registers can be listed in ascending order in the format: R?[-R[/R?[-R?...]...]] Replace the R in the above format with a register reference. Any of the following mnemonics are valid: A0-A7 D0-D7 R0-R15 The following example illustrates a sample register list. A2-A4/A7/D1/D3-D5 You can also use commas to separate registers as follows: A1,A2,D5,D7
.section #	The section directive defines a base segment. The sections can be numbered from 0 to 15 inclusive. Section 14 always maps to data. Section 15 is bss. All other section numbers denote text sections.
.text	The text directive instructs AS68 to change the assembler base segment to the text segment. Each assembly of a program begins with the first word in the text segment.

5.5 SAMPLE COMMANDS INVOKING AS68

```
(a)as68 -u -l test.s
```

This command assembles the source file TEST.S and produces the object file TEST.O. Error messages appear on the screen. Any undefined symbols are treated as global.

```
(a)as68 -p  smpl.s > smpl.l
```

This command assembles the source file SMPL.S and produces the object file SMPL.O. The program must run in the first 64K of memory; that is, no address can be larger than 16 bits. Error messages and the listing are directed to the file SMPL.L.

5.6 ASSEMBLY LANGUAGE DIFFERENCES

The syntax differences between the AS68 assembly language and Motorola's assembly language are described in the following list.

* In AS68, all assembler directives are optionally preceded by a period (.). For example,

  .equ or equ
  .ds or ds

* AS68 does not support, but accepts and ignores the following Motorola directives:

  comline
  mask2
  idnt
  opt

* The Motorola .set directive is implemented as the equate directive (equ).

* AS68 accepts upper- and lowercase characters. You can specify instructions and directives in either case. However, labels and variables are case-sensitive. For example, the label START and Start are not equivalent.

* For AS68, all labels must terminate with a colon (:). For example,

  A:
  FOO:

  However, if a label begins in column 1, it need not terminate with a colon.

* For AS68, ASCII string constants can be enclosed in either single or double quotes. For example,

  'ABCD'
  "ac14"

* For AS68, registers can be referenced with the following mnemonics:

  r0-r15
  R0-R15
  d0-d7
  D0-D7
  a0-a7
  A0-A7

  Upper- and lowercase references are equivalent. Registers R0-R7 are the same as D0-D7 and R8-R15 are the same as A0-A7.

* Use caution when manipulating the location counter forward in AS68. An expression can move the counter forward only. The unused space is filled with zeros in the text or data segments.

* For AS68, comment lines can begin with an asterisk followed by an equals sign (* =), but only if one or more spaces exist between the asterisk and the equals sign as follows:

  * = This command loads R1 with zeros.
  * = Branch to subroutine XYZ.

  Be sure to include a space after the asterisk, as the location counter is manipulated with a statement of the form:

  *=expr

* For AS68, the syntax for short form branches is bxx.b rather than bxx.s

* The Motorola assembler supports a programming model in which a program consists of a maximum of 16 separately relocatable sections and an optional absolute section. The AS68 distributed with GEMDOS does not support this model. Instead, AS68 supports a model in which a program contains three segments, text, data, and bss as described in Section 3, "Command File Format."

5.7 ASSEMBLY LANGUAGE EXTENSIONS

The following enhancements have been added to AS68 to make the assembly language more efficient:

* When the instructions add, sub, and cmp are used with an address register in the source or destination, they generate adda, suba, and cmpa. When the clr instruction is used with an address register (Ax), it generates sub Ax, Ax.

* add, and, cmp, eor, or, sub are allowed with immediate first operands and generate addi, andi, cmpi, eori, ori, and subi instructions if the second operand is not register-direct.

* All branch instructions generate short relative branches where possible, including forward references.

* Any shift instruction with no shift count specified assumes a shift count of one. For example, asl rl is equivalent to asl #1,rl.

* A jsr instruction is changed to a bsr instruction if the resulting bsr instruction is shorter than the jsr instruction.

* The .text directive causes the assembler to begin assembling instructions in the text segment. The .data directive causes the assembler to begin assembling initialized data in the data segment.

* The .bss directive instructs the assembler to begin defining storage in the bss. No instructions or constants can be placed in the bss because the bss is for uninitialized data only. However, the .ds directives can be used to define storage locations, and the location counter (*) can be incremented.

* The .globl directive in the form:

  .globl label[,label] ...

  makes the labels external. If they are otherwise defined (by assignment or appearance as a label), they act within the assembly exactly as if the .globl directive were not given. However, when linking this program with other programs, these symbols are available to other programs. Conversely, if the given symbols are not defined within the current assembly, the linker can combine the output of this assembly with that of others which define the symbols.

* The common directive (comm) defines a common region, which can be accessed by programs that are assembled separately. The syntax for the common directive is

  .comm label, expression

  The expression specifies the number of bytes allocated in the common region. If several programs specify the same label for a common region, the size of the region is determined by the value of the largest expression.

  The common directive assumes the label is an undefined external symbol in the current assembly. However, the linker, LO68, is special-cased, so all external symbols, which are not otherwise defined, and which have a non-zero value, are defined to be in the bss, and enough space is left after the symbol to hold expression bytes. All symbols which become defined in this way are located before all the explicitly defined bss locations.

* The .even directive causes the location counter (*), if positioned at an odd address, to be advanced by one byte so the next statement is assembled at an even address.

* The instructions move, add, and sub, specified with an immediate first operand and a data (D) register as the destination, generate Quick instructions, where possible.

5.8 ERROR MESSAGES

Appendix D lists the error messages generated by AS68.

5.9 INSTRUCTION SET SUMMARY

This section contains two tables that describe the assembler instruction set distributed with GEMDOS. Table 5-3 summarizes the assembler (AS68) instruction set. Table 5-4 lists variations on the instruction set listed in Table 5-3. For details on specific instructions, refer to Motorola's 16-Bit Microprocessor User's Manual, third edition, MC68000UM(AD3).

Table 5-3. Instruction Set Summary

Instruction	Description
abcd	Add Decimal with Extend
add	Add
and	Logical AND
asl	Arithmetic Shift Left
asr	Arithmetic Shift Right
bcc	Branch Conditionally
bchg	Bit Test and Change
bclr	Bit Test and Clear
bra	Branch Always
bset	Branch Test and Set
bsr	Branch to Subroutine
btst	Bit Test
chk	Check Register Against Bounds
clr	Clear Operand
cmp	Compare
dbcc	Test Condition, Decrement, and Branch
divs	Signed Divide
divu	Unsigned Divide
eor	Exclusive OR
exg	Exchange Registers
ext	Sign Extend
illegal	Illegal Instruction
jmp	Jump
jsr	Jump to Subroutine
lea	Load Effective Address
link	Link Stack
lsl	Logical Shift Left
lsr	Logical Shift Right
move	Move
movem	Move Multiple Registers
movep	Move Peripheral Data
muls	Signed Multiply
mulu	Unsigned Multiply

Table 5-3. (continued)

Instruction	Description
nbcd	Negate Decimal with Extend
neg	Negate
nop	No Operation
no	One s Complement
or	Logical OR
pea	Push Effective Address
reset	Reset External Devices
rol	Rotate Left without Extend
ror	Rotate Right without Extend
roxl	Rotate Left with Extend
roxr	Rotate Right with Extend
rte	Return From Exception
rtr	Return and Restore
rts	Return from Subroutine
sbcd	Subtract Decimal with Extend
scc	Set Conditional
stop	Stop
sub	Subtract
swap	Swap Data Register Halves
tas	Test and Set Operand
trap	Trap
trapv	Trap on Overflow
tst	Test
unlk	Unlink

Table 5-4. Variations of Instruction Types

Instruction		Variation
add	add	Add
	adda	Add Address
	addq	Add Quick
	addi	Add Immediate
	addx	Add with Extend
and	and	Logical AND
	andi	AND Immediate
	andi to ccr	
	andi to sr	
cmp	cmp	Compare
	cmpa	Compare Address
	cmpm	Compare Memory
	cmpi	Compare Immediate
eor	eor	Exclusive OR
	eori	Exclusive OR Immediate
	eori to ccr	
	eori to sr	
move	move	Move
	movea	Move Address
	moveq	Move Quick
	move to ccr	
	move to sr	
	move from sr	
	move to usp	
neg	neg	Negate
	negx	Negate with Extend
or	or	Logical OR
	ori	OR Immediate
	ori to ccr	
	ori to sr	OR Immediate to Status Register
sub	sub	Subtract
	suba	Subtract Address
	subi	Subtract Immediate
	subq	Subtract Quick
	subx	Subtract with Extend

End of Section 5

GEM DOS LINKERS

Two linkers are provided with GEM DOS: LINK68 and LO68. Both create relocatable command files in the format required by the operating system.

Note that both LINK68 and LO68 produce files in CP/M-68K relocatable format. Use the RELMOD format conversion utility to translate linker output files from CP/M-68K format to the GEM DOS relocatable format. RELMOD is described in Section 6.3.

6.1 LINK68

LINK68 is a linkage editor that combines object files into a command file. LINK68 accepts object files produced by other language processors, such as AS68 or the C compiler. LINK68 produces an executable file in either contiguous or noncontiguous command file format. Noncontiguous command files are not supported by GEM DOS. See Section 3 for a description of the contiguous file format.

LINK68 resolves all references to external symbols and concatenates the object files in the order specified. The entry point of the resulting command file is the first instruction in the first object file.

Note: You must use LINK68 to create an executable command file even when a single object file contains no unresolved references.

If you use the AS68 common directive to specify a common area shared by separate modules, LINK68 resolves all common areas with the same name to the same address in the bss segment. If more than one file specifies static storage with the same name, LINK68 uses the largest size for allocation.

6.1.1 Invoking LINK68

To invoke LINK68, enter a command of the following form:

```
LINK68 [file =] object-file-1 [,object-file-2,...object-file-n]
```

If you invoke LINK68 without any command tail, the linker lists the options, and returns to the operating system.

The first file specification is the name of the command file you want to create, and object-file-1 through object-file-n are the object files to link. For example, the following command creates the output file MATH:

```
(a)link68 math = sin,cos,tan
```

If you omit the filename to the left of the equal sign, LINK68 creates the output file using the first filename in the command line, and assigns the default filetype 68K. For example, the following command creates SIN.68K:

```
(a)link68 sin,cos,tan
```

6.1.2 LINK68 Command-Line Options

When you invoke LINK68, you can specify command-line options that control the link operation. There are two kinds of options: global and local. Global options apply to the entire link operation. Local options apply only to the individual files being linked. You enclose global options in square brackets immediately preceding the output filename (if specified) in the command line. You enclose local options in square brackets immediately following the filename to which they apply.

You can use spaces between filenames to improve readability in the command line, and you can put more than one option in square brackets by separating them with commas. LINK68 also allows you to abbreviate an option name to its shortest unambiguous form. Table 6-1 lists the LINK68 options and explains their use. Abbreviated forms are in parentheses.

Note: GEM DOS requires that the linked file have the magic number 601AH. LINK68 produces the output file with this magic number only when you request a relocatable file with contiguous text, data, and bss segments. Consequently, to create a command file to run under GEM DOS, do not select the ABSOLUTE, DATABASE[n], BSSBASE[n], or TEXTBASE[n] options when you invoke LINK68.

Table 6-1. LINK68 Command-line Options

Option	Function
ABSOLUTE (AB)	Tells LINK68 to generate an absolute file with no relocation bits. The default is a relocatable program. Note that command files must be relocatable to run under GEM DOS.
ALLMODS (AL)	Tells LINK68 to load all modules from a library, even if they are not referenced. The default is to include only those modules that are actually referenced.
BSSBASE[n] (B[n])	Sets the base address for the uninitialized data segment (bss) in discontiguous programs. The n is a hexadecimal value. The default value is the first even word after the data segment.

Table 6-1. (continued)

Option	Function
COMMAND (C)	Tells LINK68 that the following named file contains the rest of the command line. LINK68 ignores the rest of the main command line. Nested command files are not allowed. The format of this option is: COMMAND [filename] where filename is the file containing the rest of the command line.
DATABASE[n] (D[n])	Specifies the base address of the data segment in discontiguous programs. The n is a hexadecimal value. The default is the first even word after the text segment.
IGNORE (IG)	Tells LINK68 to ignore 16-bit address overflow.
INCLUDE (IN)	Tells LINK68 to load an unreferenced module from a library. The format for this option is the following, where module-name is the module you want to load: filename [INCLUDE [module-name]]
LOCALS (L)	Tells LINK68 to put local symbols in the symbol table. The default is no local symbols. LOCALS only applies from the point in the command line that it appears. The NOLOCALS option turns this option off. Use LOCALS and NOLOCALS in combination to put local symbols from specific files in the symbol table. LINK68 always ignores local symbols starting with L.
NOLOCALS (NO)	See LOCALS.
SYMBOLS (S)	Tells LINK68 to put the symbol table in the output file. The default is no symbol table in the output file.
TEMPFILES[d:] (TEM[d:])	Tells LINK68 to use drive d for temporary files. The default is the currently logged-in drive. If you use TEMPFILES, it must precede any input files on the command line.

Table 6-1. (continued)

Option	Function
TEXTBASE[n] (TEX[n])	Specifies the base address for the text segment. The n is a hexadecimal value. The default is 0H.
UNDEFINED (U)	Tells LINK68 to ignore the presence of undefined symbols in the input files. LINK68 lists the undefined symbols, then continues processing. The default action is to list any undefined symbols, then stop processing.

The following are examples of LINK68 command lines. Addresses are in hexadecimal. The first command line example links the files FOOMAIN and FOOLIB into a command file named FOOBAZ. It also tells LINK68 to include the symbol table in FOOBAZ, and place the temporary files on drive B.

```
(a)link68 [sym, tem[b:]] foobaz = foomain, foolib
```

The next example tells LINK68 to read the command line from the file LINKIT.INP. Note that closing brackets are not needed.

```
(a)link68 [com[linkit.inp
```

The file LINKIT.INP might contain the following commands:

```
(a)link68 [ab, tex[500], d[2a00], b[3000]] screen =
   scrns1 [1], iolib[a1]
```

This command creates the file SCREEN from the files SCRNS1 and IOLIB. The command tells LINK68 to create SCREEN as an absolute file with the text segment starting at 500H, the data segment starting at 2A00H, and the initialized data segment starting at 3000H. It also tells LINK68 to include local symbols from SCRNS1 and all the modules in IOLIB.

6.1.3 REDIRECTING DIAGNOSTIC OUTPUT

Normally, LINK68 sends all diagnostic output to the console. However, you can redirect this output by using the > character in the command line. For example, the following command creates MYFILE.68K on drive A, using drive B for the temporary files, and sends the diagnostics output to the file LNKMSGS.TXT on drive D:

```
(a)link68 [tem[b:] myfile.68k = moda, modb >d:lnkmsgs.txt
```

6.2 LO68

The LO68 linker combines AS68 assembled (object) programs to produce a relocatable command file. All external references are resolved. The linker must be used to create executable programs, even when a single object program contains no unresolved references. The argument routines are concatenated in the order specified. The entry point of the output is the first instruction of the first routine.

Note: GEM DOS requires that the linked file have the magic number 601AH. LO68 produces the output file with this magic number only when you request a relocatable file with contiguous text, data, and bss segments. Consequently, to create a command file to run under GEM DOS, do not select the -T, -Z, -D, -B, and -S options and select the -R option when you invoke LO68.

Appendix D lists the error messages that LO68 displays.

6.2.1 Invoking LO68

Invoke LO68 by entering a command of the following form. Table 6-2 describes the options.

```
LO68 [-F pathname] [-R] [-S] [-I] [-Uname]
   [-O filename] [-X] [-Zaddress]
   [-Daddress] [-Baddress] object filename [object filename]
   [>message filename]
```

Table 6-2. LO68 Options

Option	Meaning
-F pathname	Specifies the path name to the directory in which temporary files are created.
-R	Preserves the relocation bits so the resulting executable program is relocatable.
-S	Strips output (if specified); the output is stripped of the symbol table and relocation bits.
-I	Does not generate 16-bit address overflow messages. When you assemble a program without the AS68 -L option, the linker might generate address overflow messages if the program contains addresses longer than 16 bits.
-Uname	Forces linking of a library module that resolves the name parameter, even if the name is not referred to by any other module being linked. Normally, library modules are only linked when they are needed to resolve references in other modules. You can use this option to create a program from a library if the module resolving the name parameter calls other modules in the library.
-O filename	Gives the object file produced by LO68 the filename that you specify following the -O option. The -O option and filename are separated by one or more spaces. If you do not specify a filename, the object file has the name C.OUT.
-X	Includes all local symbols in the symbol table except those that begin with the letter L. If not specified, LO68 puts only global symbols in the symbol table. This option allows you to discard compiler internally generated labels that begin with the letter L while retaining symbols local to routines.

-Taddress*
-Zaddress*

The -T and -Z options are equivalent. The hexadecimal address given is defined by LO68 as the beginning address for the text segment. This address defaults to zero, or it can be specified as any even hexadecimal number between 0 and FFFFFFFF. This option is useful for putting programs in ROM. Hexadecimal characters can be in uppercase or lowercase.

Table 6-2. (continued)

Option	Meaning
-Daddress*	The hexadecimal address given is defined by LO68 as the beginning address for the data segment. This address defaults to the next byte after the end of the text segment, or it can be specified as any even hexadecimal number between 0 and FFFFFFFF. This option is especially useful for for putting programs in ROM. Hexadecimal address characters can be in uppercase or lowercase.
-Baddress*	The hexadecimal address given is defined by LO68 as the beginning address for the bss. This address defaults to the next byte after the end of the data segment, or it can be specified as any even hexadecimal number between 0 and FFFFFFFF.
object filename [object filename]	The name of one or more object files produced by the assembler AS68. These are the object files that LO68 links.
>message filename	If specified, error messages produced by LO68 are redirected to the file that you specify immediately after the greater-than sign (>). If you do not specify a filename, error messages are written to the standard default output device, which typically is your console terminal.

6.2.2 Sample Commands Invoking LO68

```
(a)lo68 -s -o test.68k test.o
```

This command links assembled file TEST.O into file TEST.68K and strips out the symbol table and relocation bits.

```
(a)lo68 -t4000 -d8000 -bc000 a.o b.o c.o
```

This command links assembled files A.O, B.O, and C.O to the default output file C.OUT. The text segment starts at location 4000H; the data segment starts at location 8000H; and the bss starts at location C000H.

```
(a)lo68 -i -o test.68k test.o test1.o > error
```

This command links assembled files TEST.O and TEST1.O to file TEST.68K. Any 16-bit address overflow errors are ignored; error messages are directed to the file ERROR.

6.3 RELMOD FORMAT CONVERSION UTILITY

LINK68 and LO68 produce files in CP/M-68K relocatable format. Use the RELMOD command to translate linker output file from CP/M-68K format to GEM DOS relocatable format.

The RELMOD command line is in the following form:

```
RELMOD inputfile outputfile
```

where inputfile is the CP/M-68K format file produced by the linker and outputfile is the name of the file translated to GEM DOS format.

RELMOD can also be used to translate GEM DOS format files back to CP/M-68K. This is done by simply specifying the GEM DOS file as the input file and the CP/M-68K file as the output file.

End of Section 6

ARCHIVE UTILITY

7.1 INTRODUCTION

The Archive utility, AR68, creates a library or replaces, adds, deletes, lists, or extracts object modules in an existing library. AR68 can be used on the C Run-time Library distributed with GEM DOS and documented in the GEM DOS Supplement to the C Language Programmer's Guide for CP/M-68K.

7.2 AR68 SYNTAX

To invoke AR68, specify the components of the following command line. Optional components are enclosed in square brackets ([]).

```
AR68 DRTWX[AV][F pathname][OPMOD]ARCHIVE
        OBMOD1[OBMOD2...][>filespec]
```

You can specify multiple object modules in a command line provided the command line does not exceed 127 bytes. The delimiter character between components consists of one or more spaces.

Table 7-1 lists and describes the components of the AR68 command line.

Table 7-1. AR68 Command Line Components

Component	Meaning
AR68	Invokes the Archive utility. However, if you specify only the AR68 command, AR68 returns the following command line syntax and system prompt. (a)ar68 usage: AR68 DRTWX[AV][F pathname] [OPMOD] ARCHIVE OBMOD1 [OBMOD2...][>filespec] (a)
DRTWX	Indicates you must specify one of these letters as an AR68 command. Each of these one-letter commands and its options are described in Section 7.4.
AV	Indicates you can specify one or both of these one-letter options. These options are described with the commands in Section 7.4.
F pathname	Specifies the path to the directory in which the temporary file created by AR68 resides. If no path name is specified, the current default directory is used. AR68 creates a temporary file called AR68.TMP that AR68 uses as a scratch pad area.
OPMOD	Indicates an object module within the library that you specify. The OPMOD parameter indicates the position in which additional object modules reside when you incorporate modules in the library and specify the A option.
ARCHIVE	File specification of the library.
OBMOD1 [OBMOD2 ...]	Indicates one or more object modules in a library that AR68 deletes, adds, replaces, or extracts.
>filespec	Redirects the output to the file specification you specify, rather than sending the output to the standard output device, which is usually the console device (CONSOLE). You can redirect the output for any of the AR68 commands described in Section 7.4.

7.3 AR68 OPERATION

AR68 sequentially parses the command line once. AR68 searches for, inserts, replaces, or deletes object modules in the library in the sequence in which you specify them in the command line. Section 7.4 describes each of the commands AR68 supports.

When AR68 processes a command, it creates a temporary file called AR68.TMP, which it uses as a scratch pad. After the operation is complete AR68 erases AR68.TMP. However, AR68.TMP is not always erased if an error occurs. If this occurs, erase AR68.TMP with the ERA command and refer to Appendix D for error messages output by AR68.

7.4 AR68 COMMANDS AND OPTIONS

This section describes AR68 commands and their options. Examples illustrate the effect and interaction between each command and the options it supports.

7.4.1 The D Command

The D command deletes from the library one or more object modules specified in the command. The D command supports the following option:

V — Lists the modules in the library and indicates which modules are retained and deleted by the D command. The V option precedes modules retained in the library with the lowercase letter c and modules deleted from the library with the lowercase letter d as follows:

```
(a)ar68 dv myrah.arc orc.o

c red.o
c blue.o
d orc.o
c white.o

(a)
```

The D command deletes the module ORC.O from the library MYRAH.ARC. In addition to listing the modules in the library, the V option indicates which modules are retained and deleted.

7.4.2 The R Command

The R command creates a library when the one specified in the command line does not exist, or replaces or adds object modules to an existing library. You must specify one or more object modules.

You can replace more than one object module in the library by specifying the module names in the command line. However, when the library contains two or more modules with the same name, AR68 replaces only the first module it finds that matches the one specified in the command line. AR68 replaces modules already in the library only if you specify their names prior to the names of new modules to be added to the library. For example, if you specify the name of a module you want replaced after the name of a module you are adding to the library, AR68 adds both modules to the end of the library.

By default, the R command adds new modules to the end of the library. The R command adds an object module to a library if:

* The object module does not already exist in the library.

* You specify the A option in the command line.

* The name of the module follows the name of a module that does not already exist in the library.

The A option indicates where AR68 adds modules to the library. You specify the relative position by including the OPMOD parameter with the A option.

The R command also supports the V option, which lists the modules in the library and indicates the result of the operation performed on the library. Both the A and V options are described below.

A The A option adds one or more object modules following the module specified in the command line:

```
(a)ar68 rav sdav.o myrah.arc work.o mail.o
c much.o
c sdav.o
a work.o
a mail.o
c less.o
```

The RAV command adds the object modules WORK.O and MAIL.O after the module SDAV.O in the library MYRAH.ARC. The V option, described below, lists all the modules in the library. New modules are preceded by the lowercase letter a and existing modules are preceded by the lowercase letter c.

V The V option lists the object modules that the R command replaces or adds.

```
(a)ar68 rv jnnk.man nail.o wrench.o
c saw.o
c ham.o
r nail.o
c screw.o
a wrench.o

(a)
```

The R command replaces the object module NAIL.O and adds the module WRENCH.O to the library JNNK.MAN. The V option lists object modules in the library and indicates which modules are replaced or added. Each object module that is replaced is preceded with the lowercase letter r and each one that is added is preceded with the lowercase letter a.

7.4.3 The T Command

The T command requests that AR68 print a table of contents or a list of specified modules in the library. The T command prints a table of contents of all modules in the library only when you do not specify names of object modules in the command line. It supports the following option.

V — The V option displays the size of each file in the table of contents as shown in the following example.

```
(a)ar68 tv wine.bad
rw-rw-rw- 0/0     6818 rose.o
rw-rw-rw- 0/0     2348 white.o
rw-rw-rw- 0/0      396 red.o

(a)
```

The T command prints a table of contents in the library WINE.BAD. In addition to listing the modules in the library, the V option requests the size of each module. The character string rw-rw-rw- 0/0 that precedes the module size is meaningless for GEM DOS. However, if the file is transferred to a UNIX... system, the character string denotes the file protection and file owner. The size specified by the decimal number that precedes the object module name indicates the number of bytes in the module.

7.4.4 The W Command

The W command writes a copy of an object module in the library to the >filespec parameter specified in the command line. This command allows you to extract a copy of a module from a library and rename the copy when you write it to another disk, as shown below. For this command, the >filespec parameter is required.

```
(a)ar68 w go.arc now.o > b:\root\newd\file.o
```

The W command writes a copy of the object module NOW.O from the library GO.ARC to the file FILE.O in the NEWD subdirectory on drive B.

7.4.5 The X Command

The X command extracts a copy of one or more object modules from a library and writes them to the default disk. If no object modules are specified in the command line, the X command extracts a copy of each module in the library. The X command supports the following option.

V — The V option lists only those modules the X command extracts from the library. It precedes each extracted module with the lowercase letter x as follows:

```
(a)ar68 xv jnnk.man saw.o ham.o screw.o
x saw.o
x ham.o
x screw.o
```

7.5 AR68 ERRORS

When AR68 incurs an error during an operation, the operation is not completed. The original library is not modified if the operation would have modified the library. Thus, no modules in the library are deleted, replaced, added, or extracted. Refer to Appendix D for error messages output by AR68.

When you specify the >filespec parameter in the command line to redirect the output and one or more errors occur, the error messages are sent to the output file. Thus, you cannot detect the errors without displaying or printing the file to which the output was sent. If the contents of the output file is an object file (see the W command), you must use the DUMP utility described in Section 8 to read any error messages.

End of Section 7

MORE PROGRAMMING UTILITIES

8.1 INTRODUCTION

This section describes the DUMP, SIZE68, SENDC68, and XREF programming utilities. DUMP displays the contents of files in hexadecimal and ASCII notation. SIZE68 displays the total size of a memory image command file and the size of each of its program segments. XREF produces a cross reference symbol table for GEM DOS object files. SENDC68 creates a file of Motorola S-records from a command file. (Refer to Appendix E for a detailed description of the S-record format.)

8.2 DUMP UTILITY

The DUMP utility (DUMP) displays the contents of a GEM DOS file in both hexadecimal and ASCII notation. You can use DUMP to display any GEM DOS file regardless of the format of its contents (binary data, ASCII text, an executable file).

8.2.1 Invoking DUMP

Invoke DUMP by entering a command with the following input components in the following format:

DUMP [-sxxxx] filename1 [>filename2]

Table 8-1 lists the DUMP command line components and their meanings.

Table 8-1. DUMP Command Line Components

Component	Meaning
-sxxxx	xxxx is an optional offset (in hexadecimal) into the file. If specified, DUMP starts dumping the contents of the file from the byte-offset xxxx and continues until it displays the contents of the entire file. By default, DUMP starts dumping the contents of the file from the beginning of the file until it dumps the contents of the entire file.
filename1	Name of the file you want to dump.
>filename2	The greater than sign (>) followed by a filename or logical device optionally redirects the output of DUMP. You can specify any valid GEM DOS specification, or one of the logical device names, CON: (console) or LST: (list device). If you do not specify this optional parameter, DUMP sends its output to the console.

8.2.2 DUMP Output

DUMP sends its output to the console (or to a file or device, if specified), 8 words per line, in the following format:

```
rrrr oo (ffffff):  hhhh hhhh hhhh hhhh hhhh hhhh hhhh hhhh
*aaaaaaaaaaaaaaaa*
```

The components of a DUMP command line are as follows:

Component	Meaning
rrrr	Record number (GEM DOS records are 128 bytes) of the current line of the display.
oo	Offset (in hex bytes) from the beginning of the GEM DOS record.
ffffff	Offset (in hex bytes) from the beginning of the file.
hhhh	Contents of the file displayed in hexadecimal.
aaaaaaaa	Contents of the file displayed as ASCII characters. If any character is not representable in ASCII, it is displayed as a period (.).

8.2.3 DUMP Examples

In the following example, DUMP is invoked to display the contents of a command file that contains data in both binary and ASCII form.

```
(a)dump dump.68k

0000 00 (000000):  601a 0000 1b34 0000 011d 0000 0e5e 0000
*`....4.......^..*
0000 10 (000010):  0000 0000 0000 0000 0900 ffff 6034 4320
*............`4C *
0000 20 (000020):  5275 6e74 696d 6520 436f 7079 7269 6768 *Run-
time Copyright*
0000 30 (000030):  7420 3139 3832 2062 7920 4469 6769 7461 *t
1982 by Digita*
0000 40 (000040):  6c20 5265 7365 6172 6368 2056 3031 2c30 *l
Research V01.0*
0000 50 (000050):  3320 206f 0004 2268 0018 2649 d3e8 001c *3
o.."h..&ISh..*

 . . . . (and so on) . . .
```

8.3 SIZE68 UTILITY

The SIZE68 utility (SIZE68) indicates if the program segments within one or more command files are contiguous or non-contiguous, displays the size of each program segment and the symbol table, and reports if the command files are relocatable or non-relocatable. SIZE68 displays both decimal and hexadecimal values for the sizes of the program segments and the symbol table. GEM DOS command files usually have a filetype of .PRG or .REL. The total size of a command file's segments returned by SIZE68 and the size of a command file returned by the DIR command are not equal. The file size returned by SIZE68 includes the size of the text, data, and bss program segments and the size of the symbol table but does not include the size of the header and and relocation bits. For more details on the DIR command, refer to the GEM DOS User's Guide.

8.3.1 Invoking SIZE68

Invoke SIZE68 by entering a command line with the following format:

SIZE68 filename [filename2 filename3, ...] [>outfile]

The SIZE68 command line components have the following meaning:

Component	Meaning
filename	File specification of a file whose size you want to determine.
filename2 filename3	One or more additional file specifications of files whose sizes you want to determine. SIZE68 can process multiple files, provided the command line does not exceed 128 bytes. Note that SIZE68 also accepts wildcard file specifications.
>outfile	Specifies the file specification to which SIZE68 sends its output. If you do not specify an output file specification, SIZE68 sends the output to the console. For the output file specification, you can specify a valid GEM DOS filename, or one of the logical device names, CON: (console) or LST: (list device).

8.3.2 SIZE68 Examples and Output

This section contains two examples that show SIZE68 command lines and output.

1. The following SIZE68 command line example returns information about the program segments in one command file.

```
(a)SIZE68 SIZE68.PRG
SIZE68.PRG:
        Contiguous
        .text length              =    9312          2460
        .data length              =    1178           49A
        .bss length               =    9140          23B4
        Symbol table length       =       0             0
        Start of .text            =       0             0
        File is relocatable.
```

SIZE68.PRG contains a 9312-byte (decimal) text segment, a 1179-byte (decimal) data segment, and a 9140-byte (decimal) bss; the segments are contiguous and SIZE68 is relocatable. Hexadecimal notations for the decimal values are displayed in the last column of SIZE68 output.

2. The following SIZE68 command line uses a wildcard file specfication to return information on an object file, a command file, and a text file.

```
(a)SIZE68 FI*.*
FIND.O:
        Contiguous
        .text length              =    1072           430
        .data length              =     188            BC
        .bss length               =       0             0
        Symbol table length       =    1708           6AC
        Start of .text            =       0             0
        File is relocatable.

FIND.PRG:
        Contiguous
        .text length              =    9888          26A0
        .data length              =    1060           424
        .bss length               =    9396          24B4
        Symbol table length       =       0             0
        Start of .text            =       0             0
        File is relocatable.

FILE.MSG:
        Not a program file.
```

Notice that when you specify a file that is not a command file (FILE.MSG, an ASCII file, for example), SIZE68 displays:

```
Not a program file.
```

When you specify an absolute program file whose segments are non-contiguous in a SIZE68 command line, SIZE68 includes the following messages in its output:

```
Non-contiguous

No relocation information in file.
```

8.4 SENDC68 UTILITY

SENDC68 creates a file with Motorola S-record format from an absolute command file. S-records are a way to represent an absolute program in ASCII character form. For a detailed description of the S-record format, refer to Appendix E.

8.4.1 Invoking SENDC68

Invoke SENDC68 by entering a command line in the following format:

SENDC68 [-] inputfile [outputfile]

The SENDC68 command line components are described below.

Component	Meaning
-	A hyphen is optional. If you specify a hyphen, SENDC68 does not create any S-records for the bss segment. The result is a smaller S-record file. If you do not specify a hyphen, SENDC68 fills the bss segment with zeros.
inputfile	File that SENDC68 converts to the S-record format. The command file must be an absolute file in the format produced by LINK68, LO68, or RELOC.
outputfile	File that SENDC68 sends the new S-record file to. If you do not specify an output file, SENDC68 sends the S-records to the console screen.

8.4.2 SENDC68 Example

The following command line example illustrates how to convert an absolute command file into a file in the Motorola S-record format. In this example, SENDC68 creates an S-record file named PROG.SR from an absolute command file named PROG.68K.

```
(a)>sendc68 - prog.68k prog.sr
```

Note that the hyphen directs SENDC68 not to create S-records for the bss segment.

8.5 XREF UTILITY

The XREF utility generates a cross-reference table of symbols for GEM DOS object files. XREF output lists the symbols, the object file in which they are defined, and the object files in which they are accessed. XREF provides a separate listing for undefined symbols and the object files that call them. XREF accepts wildcards in the object file specifications.

8.5.1 Invoking XREF

To use XREF, enter a command of the following format:

XREF objectfile1 [objectfile2 ... objectfilelast]

Where "objectfilei" is the name of the object file for whose symbols you want a cross-referenced table. Multiple object file names may be specified.

8.5.2 XREF Examples

This section contains two example XREF command lines and shows the format of XREF output.

1. The following XREF command generates a cross-referenced table of symbols for the file GSXVAR.O.

```
(a)XREF GSXVAR.O

Block Data:
 Symbol          Defined In    Accessed In

cur_ms_s          GSXVAR
disab_cn          GSXVAR
draw_fla          GSXVAR
.                                        .
.                                        .
.                                        .
_angle            GSXVAR       MONOBJ    MONOUT
_beg_ang          GSXVAR       MONOBJ    MONOUT
.                                        .
.                                        .
.                                        .


Undefined External References:
 Symbol         Defined In   Accessed In

_ABLINE          ********   MONOUT
_CHUP            ********   MONOUT      MONOBJ
.                                            .
.                                            .
.                                            .
```

2. The example XREF command shown below uses a wildcard file specification to generate a cross-referenced table of symbols for each object file in the current directory.

```
(a)XREF *.O

Functions:
 Symbol        Defined In   Accessed In

_arb_cor        MONOBJ       MONOUT
_arrow          MONOUT
_Calc_pt        MONOUT
.                                    .
.                                    .
.                                    .
```

End of Section 8

SID68 DEBUGGER

SID68 allows you to test and debug programs interactively in the GEM DOS environment. The presentation of information in this section assumes you are familiar with the MC68000 microprocessor, the assembler (AS68), and the GEM DOS operating system.

9.1 INVOKING SID68

Invoke SID68 by entering one of the following commands:

```
SID
SID filename
```

The first command loads and executes SID68. After it is loaded, SID68 displays its sign-on message and the hyphen (-) prompt character to show it is ready to accept commands.

The second command invokes SID68 and loads the file specified by filename. If the filetype is not specified, it defaults to the 68K filetype. The second form of the command is equivalent to the following sequence in which the first command is issued and then, at the SID68 prompt, the E command is issued to load a file for execution under SID68:

```
(a)SID

SID68
Copyright 1982, Digital Research

-Efilename
```

9.1.1 SID68 Command Conventions

When SID68 is ready to accept a command, it prompts you with a hyphen (-). In response, you can type a command line or a Ctrl-C (^C) to end the debugging session. A command line can have as many as 64 characters, and must be terminated with a Return. When entering the command, use standard GEM DOS line-editing functions to correct typing errors. SID68 does not process the command line until you enter a Return.

Table 9-1 summarizes SID68 commands. They are defined individually later in this section.

Table 9-1. SID68 Command Summary

Command	Action
D	Display memory in hexadecimal and ASCII.
E	Load program for execution.
F	Fill memory block with a constant.
G	Begin execution with optional breakpoints.
H	Use hexadecimal arithmetic.
I	Set up file control block and command tail.
L	List memory using MC68000 mnemonics.
M	Move memory block.
P	Set and remove permanent breakpoints.
R	Read disk file into memory.
S	Set memory to new values.
T	Trace program execution.
U	Untrace program monitoring.
V	Show memory layout of disk file read.
W	Write contents of memory block to disk.
X	Examine and modify CPU state.

The command character can be followed by one or more arguments, which can be hexadecimal values, filenames, or other information, depending on the command. Some commands can operate on byte, word, or longword data. The letters W for word or L for longword must be appended to the command character for commands that operate on multiple data lengths. Arguments are separated from each other by commas or spaces.

See Section 9.2 for more details on each command.

9.1.2 Address Specifications

Most SID68 commands require one or more addresses as operands. All addresses are entered as hexadecimal numbers of up to eight hexadecimal digits (32 bits).

9.1.3 Symbol References

SID68 allows you to reference symbols in place of addresses. You can display the symbols, along with the other disassembled instructions in your executable file, by using the L command described in Section 9.2.7. When entering a command that specifies a symbol, precede the symbol name with a period as follows:

```
command.symbol
```

For example, to direct the GO command to execute from the current Program Counter (PC) to the symbol QUIT in the object file, enter:

```
g,.quit
```

The C compiler puts an underscore (_) at the beginning of external symbols. For example, to specify the GO command with a breakpoint at the C function BLIVOT, enter:

```
g,._blivot
```

9.1.4 Stopping SID68

Stop SID68 by typing a ^C in response to the hyphen prompt. This returns control to the CCP.

9.1.5 SID68 Operation with Interrupts

SID68 operates with interrupts enabled or disabled, and preserves the interrupt state of the program being executed under SID68. When SID68 has control of the CPU, either when it is initially invoked or when it regains control from the program being tested, the condition of the interrupt mask is the same as it was when SID68 was invoked, except for a few critical regions where interrupts are disabled. While the program being tested has control of the CPU, the user's CPU state, which can be displayed with the X command, determines the state of the interrupt mask.

Note that SID68 uses the Trace and Illegal Instruction exceptions. Therefore, programs debugged under test should not use these.

9.2 SID68 COMMANDS

This section defines SID68 commands and their arguments. SID68 commands allow you to control program execution and display and modify system memory and the CPU state.

9.2.1 The D (Display) Command

The D command displays the contents of memory as 8-bit, 16-bit, or 32-bit hexadecimal values and in ASCII. The forms are

```
D
Ds
Ds,f
DW
DWs
DWs,f
DL
DLs
DLs,f
```

where s is the starting address, and f is the last address that SID68 displays.

Memory is displayed on one or more lines. Each line shows the values of up to 16 memory locations. For the first three forms, the display line appears as follows:

```
aaaaaaaa bb bb ... bb cc ... cc
```

where aaaaaaaa is the address of the data being displayed. The bb's represent the contents of the memory locations in hexadecimal, and the cc's represent the contents of memory in ASCII. Any non-graphic ASCII characters are represented by periods.

In response to the Ds form of the D command, shown above, SID68 displays 12 lines that start from the current address.

Form Ds,f displays the memory block between locations s and f. Forms DW, DWs, and DWs,f are identical to D, Ds, and Ds,f except the contents of memory are displayed as 16-bit or word values, as shown below:

```
aaaaaaaa  wwww  wwww  ...  wwww  cccc  ...  cccc
```

Forms DL, DLs, and DLs,f are identical to D, Ds, and Ds,f except the contents of memory are displayed as 32-bit or longword values, as shown below:

```
aaaaaaaa llllllll llllllll ... llllllll cccccccc ...
```

During a display, you can abort the D command by typing any character at the console.

9.2.2 The E (Load for Execution) Command

The E command loads a file in memory so that a subsequent G, T, or U command can begin program execution. The syntax for the E command is

```
Efilename
```

where filename is the name of the file to be loaded. If no filetype is specified, the filetype 68K is assumed.

An E command reuses memory used by any previous E command. Thus, you can load only one file at a time for execution.

When the load is complete, SID68 displays the starting and ending addresses of each segment in the file. Use the V command to display this information at a later time.

If the file does not exist or cannot be successfully loaded in the available memory, SID68 displays an error message. See Appendix A for error messages returned by SID68.

9.2.3 The F (Fill) Command

The F command fills an area of memory with a byte, word, or longword constant. The forms are

```
Fs,f,b
FWs,f,w
FLs,f,l
```

where s is the starting address of the block to be filled, and f is the address of the final byte of the block within the segment specified in s.

In response to the first form, SID68 stores the 8-bit value b in locations s through f. In the second form, the 16-bit value w is stored in locations s through f in standard form: the high 8 bits are first, followed by the low 8 bits. In the third form, the 32-bit value l is stored in locations s through f with the Most Significant Byte first.

If s is greater than f, SID68 responds with a question mark. Also, if b is greater than FF hexadecimal (255), w greater than FFFF hexadecimal (65,535), or l greater than FFFFFFFF hexadecimal (4,294,967,295), SID68 responds with a question mark. SID68 displays an error message if it cannot read back the value stored in memory successfully. This error indicates a faulty or non-existent RAM location.

9.2.4 The G (Go) Command

The G command transfers control to the program being tested, and optionally sets one to ten breakpoints. The forms are as follows:

```
G
G,b1,...b10
Gs
Gs,b1,...b10
```

where s is the address at which the program begins executing and b1 through b10 are addresses of breakpoints.

In the first two forms, no starting address is specified. SID68 starts executing the program at the address specified by the Program Counter (PC). The first form transfers control to the program without setting any breakpoints. The second form sets breakpoints before passing control to the program. The last two forms are analogous to the first two except that the PC is first set to s.

Once control has been transferred to the program under test, it executes in real-time until it encounters a breakpoint. At this point, SID68 regains control, clears all breakpoints, and displays the CPU state in the same form as the X command. When a breakpoint returns control to SID68, the instruction at the breakpoint address has not yet been executed. To set a breakpoint at the same address, you must specify a T or U command first.

9.2.5 The H (Hexadecimal Math) Command

The H command computes the sum and difference of two 32-bit values. The form is

```
Ha,b
```

where a and b are the values whose sum and difference SID68 computes. SID68 displays the sum (ssssssss) and the difference (dddddddd) truncated to 16 bits as follows:

```
ssssssss dddddddd
```

9.2.6 The I (Input Command Tail) Command

The I command prepares a file control block (FCB) and command tail buffer in the base page of the last file loaded with the E command. The form is as follows:

```
Icommandtail
```

where commandtail is the character string which usually contains one or more filenames. The first filename is passed into the default File Control Block at 005CH. The optional second filename, if specified, is passed into the second default File Control Block beginning at 0038H. The characters in the command tail are also copied to the default command buffer at 0080H. The length of the command tail is stored at 0080H, followed by the character string terminated with a binary zero.

If a file has been loaded with the E command, SID68 copies the File Control Block and command buffer from the base page of SID68 to the base page of the program loaded.

9.2.7 The L (List) Command

The L command lists the contents of memory in assembly language. The forms are as follows:

```
L
Ls
Ls,f
```

where s is the starting address, and f is the last address in the list.

The first form lists 12 lines of disassembled machine code from the current address. The second form sets the list address to s and then lists 12 lines of code. The last form lists disassembled code from s through f. In all three cases, the list address is set to the next unlisted location in preparation for a subsequent L command. When SID68 regains control from a program being tested (see G, T, and U commands), the list address is set to the address in the Program Counter (PC).

Long displays can be aborted by pressing any key during the list process. Or, enter Ctrl-S (^S) to halt the display temporarily. A Ctrl-Q (^Q) restarts the display after ^S halts it.

The syntax of the assembly language statements produced by the L command is described in the Motorola 16-Bit Microprocessor User's Manual, third edition, MC68000UM(AD3). Section 9.2.17 describes some minor differences between the assembly language statements produced by the L command and standard Motorola 68000 assembly language.

9.2.8 The M (Move) Command

The M command moves a block of data values from one area of memory to another. The form is as follows:

```
Ms,f,d
```

where s is the starting address of the block to be moved, f is the address of the final byte of the block to be moved, and d is the address of the first byte of the area to receive the data. Note that if d is between s and f, part of the block being moved will be overwritten before it is moved, because data is transferred starting from location s.

9.2.9 The P (Pass Points) Command

The P command sets, clears, and displays pass points. The forms are as follows:

```
Pa,n
Pa
-P
```

A pass point is a permanent breakpoint that remains in effect until you explicitly remove it, as opposed to breakpoints set with the G command that must be reentered with each G command. Pass points have associated pass counts ranging from 1 to OFFFFH. The pass count indicates how many times the instruction at the pass point executes before the control returns to the console. SID68 can set up to 16 pass points at a time.

An important distinction between breakpoints and pass points is that when execution stops at a breakpoint, the instruction at the breakpoint has not been executed. When execution stops due to a pass point whose pass count has reached 1, the instruction at the pass point has been executed. This makes it simple to proceed from a pass point with a G command without encountering the same pass point.

Forms Pa,n and Pa set pass points. Form Pa,n sets a pass point at address a (pass point address) with a pass count of n (from 1 to OFFFH). If a pass point is already active at a, the pass count is changed to n. SID68 responds with a question mark if there are already 16 active pass points.

Form Pa sets a pass point at address a with a pass count of 1. If a pass point is already active at address a, the pass count is changed to 1 if it is not 1. SID68 responds with a question mark if there are already 16 active pass points.

The -P form is used to clear pass points.

9.2.10 The R (Read) Command

The R command reads a file to a contiguous block in memory. The format is

```
Rfilename
```

where filename is the name and type of the file to be read.

SID68 reads the file into memory and displays the starting and ending addresses of the block of memory occupied by the file. A Value (V) command can redisplay the information at a later time. The default display pointer, for subsequent Display (D) commands is set to the start of the block occupied by the file.

9.2.11 The S (Set) Command

The S command can change the contents of bytes, words, or longwords in memory. The forms are

```
Ss
SWs
SLs
```

where s is the address at which the change is to occur.

SID68 displays the memory address and its current contents. In response to the first form, the display is

```
aaaaaaaa bb
```

In response to the second form, the display is

```
aaaaaaaa wwww
```

In response to the third form, the display is

```
aaaaaaaa llllllll
```

where bb, wwww, and llllllll are the contents of memory in byte, word, and longword formats, respectively.

In response to one of the above displays, you can alter the memory location or leave it unchanged. If you enter a valid hexadecimal value, the contents of the byte, word, or longword in memory is replaced with that value. If you do not enter a value, the contents of memory are unaffected and the contents of the next address are displayed. In either case, SID68 continues to display successive memory addresses and values until either a period or an invalid value is entered.

SID68 displays an error message if it cannot read back the value stored in memory successfully. This error indicates a faulty or non-existent RAM location.

9.2.12 The T (Trace) Command

The T command traces program execution for 1 to OFFFFFFFFH program steps. The forms are

```
T
Tn
Tw
```

where n is the number of instructions to execute before returning control to the console.

After SID68 traces each instruction, it displays the current CPU state and the disassembled instruction in the same form as the X command display.

Control transfers to the program under test at the address indicated in the program counter. If you do not specify n, one instruction is executed. Otherwise, SID68 executes n instructions and displays the CPU state before each step. You can abort a long trace before all the steps have been executed by pressing any character at the console.

The Tw form traces execution without calls to subroutines. The entire subroutine called from the program level being traced is treated as a single program step and executed in real time. This allows tracing at a high level of the program, ignoring subroutines that are already debugged.

After a T command, the list address used in the L command is set to the address of the next instruction to be executed.

Note that SID68 does not trace through a BDOS interrupt instruction since SID68 itself makes BDOS calls and the BDOS is not reentrant. Instead, the entire sequence of instructions from the BDOS interrupt through the return from BDOS is treated as one traced instruction.

9.2.13 The U (Untrace) Command

The U command is identical to the Trace (T) command except that the CPU state is displayed only after the last instruction is executed, rather than after every step. The forms are

```
U
Un
```

where n is the number of instructions to execute before control returns to the console. You can abort the command before all the steps have been executed by pressing any key at the console.

9.2.14 The V (Value) Command

The V command displays information about the last file loaded with the Load For Execution (E) or Read (R) commands. The form is

```
V
```

If the last file was loaded with the E command, the V command displays the starting address and length of each of the segments contained in the file, the base page pointer, and the initial stack pointer. The format of the display is

```
Text base=00000500 data base=00000B72 bss base=00003FDA
text length=00000672 data length=00003468 bss length=0000A1B0
base page address=00000400  initial stack pointer=000066D4
```

If no file has been loaded, SID68 responds to the V command with a question mark.

9.2.15 The W (Write) Command

The W command writes the contents of a contiguous block of memory to disk. The forms are

```
Wfilename
Wfilename,s,f
```

The filename is the file specification of the disk file that receives the data.

If you use the first form, SID68 assumes the values for s and f from the last file read with an R command. If no file has been read by an R command, SID68 responds with a question mark. This form is useful for writing out files after patches have been installed, assuming the overall length of the file is unchanged.

In the second form the letters s and f are the first and last addresses of the block to be written. If f does not specify the last address, SID68 uses the same value that was used for s.

If the file specified in the W command already exists on disk, SID68 deletes the existing file before it writes the new file.

9.2.16 The X (Examine CPU State) Command

The X command displays the entire state of the CPU, including the Program Counter (PC), User Stack Pointer (USP), System Stack Pointer (SSP), Status Register (ST, displayed by field), all eight data registers, all eight address registers, and the disassembled instruction at the memory address currently in the PC. The forms are

```
X
Xr
```

where r is one of the following registers:

D0 to D7, A0 to A7, PC, USP, or SSP

The first form displays the CPU state as follows:

```
PC=00016000 USP=00001000 SSP=00002000 ST=FFFF=> (etc.)
D  00001000 00000D01 ... 00000001
A  000B0A00 000A0010 ... 00000000
lea $16028,A0
```

The first line includes:

PC	program counter
USP	user stack pointer
SSP	system stack pointer
ST	status register

Following the status register contents on the first display line, you see the values of each bit in the status register, as shown in the following sample:

```
TR SUP IM=7 EXT NEG ZER OFL CRY
```

This sample display includes:

TR	Trace Bit
SUP	Supervisor Mode Bit
IM=7	Interrupt Mask=7
EXT	Extend
NEG	Negative
ZER	Zero
OFL	Overflow
CRY	Carry

The second form, Xr, allows you to change the value in the registers of the program being tested. The r identifies the register. SID68 responds by displaying the current contents of the register, leaving the cursor on that line. If you type a Return, the value is not changed. If you type a new valid value and then a Return, the register is changed to the new value. You can change the contents of all registers except the status register.

9.2.17 Assembly Language Syntax for the L Command

In general, the syntax of the assembly language statements used in the L command is standard Motorola 68000 assembly language. Several minor exceptions are given in the following list:

* SID68 prints all numeric values in hexadecimal.

* SID68 uses lowercase mnemonics.

* SID68 assumes word operations unless a byte or longword specification is explicitly stated.

End of Section 9

SUMMARY OF BIOS FUNCTIONS

Table A-1 lists the BIOS functions supported by GEM DOS.

Table A-1. Summary of BIOS Functions

00 Get Memory Parameter Block

```
VOID        Get_MPB(p_MPB)
MPB         *p_MPB;
{
            Fill in Memory Parameter Block at p_MPB.
}
```

01 Character input status

```
LONG     Char_In_Stat(handle)
WORD     handle;
{
         if (device is ready)
                     return (-1);
         return (0);
}
```

02 Character input

```
LONG     Char_Input(handle)
WORD     handle;
{
         return (character for device 'handle');
```

03 Character output

```
VOID     Char_output(handle, char)
WORD     handle, char;
{
         Output 'char' to device 'handle'.  Character may be
         output directly for polled output, or enqued for
         interrupt driven output.
}
```

Table A-1. (continued)

04 Read/Write disk sectors

```
LONG    Read_Write_sectors(wrtflg, buffer, count, begin, drive)
WORD    wrtflg;             /* 0: read, 1: write */
BYTE    *buffer;            /* address of transfer buffer */
WORD    count;              /* number of sectors to transfer */
WORD    begin;              /* beginning sector number */
WORD    drive;              /* 0: A, 1: B, ... */
{
        Perform indicated disk operation.
        return (error indication or E_OK);
}
```

05 Get/Set exception vector

```
LONG    Get_Set_Vector(n, addr)
WORD    n;                  /* vector number for operation  */
LONG    addr;               /* address/parameter value      */
{
        temp = vector[n];
        if (addr != -1L)
                vector[n] = addr;
        return (temp);
}
```

Extended vectors

0x101 Critical error handler

It is the responsibility of the BIOS to invoke the routine at this vector whenever a disk related critical error occurs, such as write fault, drive not read, etc.

It is the responsibility of the handler that installs itself to save d3-d7/a3-a7 if they are used. The error number of the critical error is a WORD parameter on the stack.

To ignore the error, set D0.L to 0 and return.

To retry, set D0.L to 0x10000 and return.

To abort, move error parameter to D0 and sign extend it to a LONG.

Table A-1. (continued)

If the target system is intended to support character-based (that is, non-GEM related) applications, it is the responsibility of the BIOS implementor to make the console I/O portion of the BIOS re-entrant from this handler so that user prompting can be done.

0x102 Terminate handler

It is the responsibility of the handler to determine whether it should allow the process termination to continue. To terminate, simply RTS. Otherwise, 'longjump' back into the main code of the process.

06 Get tick info

```
LONG    get_ticks()
{
        return (number of milliseconds per tick);
}
```

07 Get BIOS Parameter Block

```
BPB     *get_bpb(drive)
WORD    drive;
{
        If multiple media types are supported on 'drive',
        determine type of media.

        return(pointer to BPB for this disk);
}
```

08 Character output status

```
LONG    Char_Out_Stat(handle)
WORD    handle;
{
        If (device is ready for output)
                    return (-1);
        return (0);
}
```

Table A-1. (continued)

09 Media change

```
LONG    Media_Change(drive)
WORD    drive;
{
        if (media has not changed)
                return (0);
        if (media may have changed)
                return (1);
        if (media has definitely changed)
                return (2);
}
```

0A Get drive map

```
LONG    Get_Drive_Map()
{
        return (bit map of accessible drives on system)
}
```

Where bit 0 represents drive A, etc.

NOTE: If the BIOS supports logical drives A and B on a single physical it should return both bits set if a floppy disk is present.

0B Get/Set shift keys (console)

```
LONG    Get_Set_Shift(flag)
WORD    flag;
{
        temp = state;
        if (flag != -1)
                state = flag;
        return (temp);
}
```

The bits in 'flag' are assigned to be:

0: Right shift key
1: Left shift key
2: Control key
3: Alt key
4: Caps lock key

Table A-1. (continued)

0C Character control input

```
LONG    Char_Ctl_In(handle, length, buffer)
WORD    handle, length;
BYTE    *buffer;
{
        Read up to 'length' bytes from 'handle's control
        channel into 'buffer.
        Called with 'length' = 0 to determine if
        device accepts control strings.
}
```

0D Character control output

```
LONG    Char_Ctl_Out(handle, length, buffer)
WORD    handle, length;
BYTE    *buffer;
{
Write 'length' bytes to 'handle's control channel from 'buffer'.
}
```

0E Disk control input

```
LONG    Disk_Ctl_In(drive, length, buffer)
WORD    drive, length;
BYTE    *buffer;
{
        Read up to 'length' bytes from 'handle's control channel
        into 'buffer'.  Called with 'length' = 0 to determine if
        'handle' accepts control strings.
}
```

0F Disk control output

```
LONG    Disk_Ctl_Out(drive, length, buffer)
WORD    drive, length;
BYTE    *buffer;
{
        Write 'length' bytes to 'handle's control
        channel from 'buffer'.
}
```

Table A-1. (continued)

10 Character vector exchange

```
LONG    Char_Vec_Exchange(handle, address)
WORD    handle;
LONG    address;
{
        temp = 'handle's old handler address;

        Install 'address' as 'vector's
        new logical interrupt handler.

        return (temp);
}
```

It is not strictly necessary to implement this function for CON: in the current release of GEM DOS.

Stack format for invocation of character vector handler

```
sp+8    LONG (see below)
sp+4    LONG flags
sp      LONG return address
```

Interpretation of bits in 'flags' LONG:

0 Packet received
If this is 1, then the LONG at sp+8 is defined as
For CON:, AUX:, or PRN:
The LONG character info that BIOS function 0x02 would have returned.
For CLOCK
The number of milliseconds since the last tick.
For MOUSE
A pointer to a parameter block that contains:
A BYTE of button status
A BYTE of delta-X
A BYTE of delta-Y

1 Error

If no other error status bit is on, this is just an error that couldn't be described any better with the available defined status bits.

2 Out of paper

3 Off line

4 Timeout

5 Framing Error

6 Parity error

7-15 RESERVED

If no packet received and no error then state changed on one of the following:

16 Carrier detect

17 Clear to send

18-31 RESERVED

Only register A7 must be preserved by a vector handler.

The handler must jump to the next handler in the chain (at the address returned by the F_IOCtl character vector exchange function), particularly for the clock tick.

System Initialization Sequence :

1. Disable interrupts if necessary.

2. Perform necessary chip and peripheral initialization to put the HW environment in a quiescent state.

3. Initialize TRAP #13 vector to the BIOS's handler address.

4. Initialize sector buffers similar to the following code:

```
char secbuf[4][512];
BCB bcbx[4];
BCB *bufl[2];
bcbx[0].b_link = &bcbx[1];
bcbx[2].b_link = &bcbx[3];
bcbx[0].b_bufdrv = -1;
bcbx[1].b_bufdrv = -1;
bcbx[2].b_bufdrv = -1;
bcbx[3].b_bufdrv = -1;
bcbx[0].b_bufr = &secbuf[0][0];
bcbx[1].b_bufr = &secbuf[1][0];
bcbx[2].b_bufr = &secbuf[2][0];
bcbx[3].b_bufr = &secbuf[3][0];
bufl[0] = &bcbx[0];       /* fat buffers */
bufl[1] = &bcbx[2];       /* dir/data buffers */
```

(The array of BCB pointers, bufl[], which has the linkable name _bufl, must be a global label. It will be linked with the GEM DOS (tm) object modules.)

5. Invoke osinit() (or '_osinit' from assembler).

6. Perform BDOS D_SetDrv(default_drive) via TRAP #1.

7. If CLI is to be executed normally (that is, loaded from disk and executed):

   Perform BDOS P_Exec(0,"command.prg","","path=default_path\0") via TRAP #1.

   Else, to execute a linked in CLI:

   * Include the modules COMA.O, and COMMAND.O, and LMUL.O when linking the system.

   * Perform BDOS P_Exec(5,"","","") via TRAP #1, which returns the address of a partially initialized base page.

   * Move the address of cli() (that is, _cli) to base page + p_tbase (8).

   * Perform BDOS P_Exec(4,"","","path=default_path\0").

Linking the CLI with the system image allows new types of media to be 'INIT'ed from the command line, provided that formatting has already been done.

Note: When the system is linked, the label _GSX_ENT will be undefined. This does not affect the operation of the system.

If the system is to be generated for 68010 (or similar) hardware, the module RWA10.O should be used instead of RWA.O.

End of Appendix A

BASE PAGE FORMAT

Table B-1 shows the format of the base page. The base page describes a program's operating environment. The Load and Execute a Process function (4B) initializes the base page when invoked to load an executable command file. For more details on the Load and execute a Process function and command files, refer to the appropriate sections of this guide.

Table B-1. Base Page Format: Offsets and Contents

Offset	
00	
04	
08	
0C	
10	
14	
18	
1C	
2C	
80	
	the program

Note that all other base page fields are reserved and not to be tampered with.

End of Appendix B

ASCII AND HEXADECIMAL CONVERSIONS

ASCII stands for American Standard Code for Information Interchange. The code contains 96 printing and 32 non-printing characters used to store data on a disk. Table C-1 defines ASCII symbols. Table C-2 shows the conversions of the ASCII character code in binary, decimal, and hexadecimal notations.

Table C-1. ASCII Symbols

Symbol	Meaning	Symbol	Meaning
ACK	acknowledge	FS	file separator
BEL	bell	GS	group separator
BS	backspace	HT	horizontal tabulation
CAN	cancel	LF	line feed
CR	carriage return	NAK	negative acknowledge
DC	device control	NUL	null
DEL	delete	RS	record separator
DLE	data link escape	SI	shift in
EM	end of medium	SO	shift out
ENQ	enquiry	SOH	start of heading
EOT	end of transmission	SP	
ESC	escape	STX	start of text
ETB	end of transmission	SUB	
ETX	end of text	SYN	synchronous idle
FF	form feed	US	unit separator
		VT	vertical tabulation

Table C-2. ASCII Conversion Table

Binary	Decimal	Hexadecimal	
0000000	0	0	NUL
0000001	1	1	SOH (CTRL-A)
0000010	2	2	STX (CTRL-B)
0000011	3	3	ETX (CTRL-C)
0000100	4	4	EOT (CTRL-D)
0000101	5	5	ENQ (CTRL-E)
0000110	6	6	ACK (CTRL-F)
0000111	7	7	BEL (CTRL-G)
0001000	8	8	BS (CTRL-H)
0001001	9	9	HT (CTRL-I)
0001010	10	A	LF (CTRL-J)
0001011	11	B	VT (CTRL-K)
0001100	12	C	FF (CTRL-L)
0001101	13	D	CR (CTRL-M)
0001110	14	E	SO (CTRL-N)
0001111	15	F	SI (CTRL-O)
0010000	16	10	DLE (CTRL-P)
0010001	17	11	DC1 (CTRL-Q)
0010010	18	12	DC2 (CTRL-R)
0010011	19	13	DC3 (CTRL-S)
0010100	20	14	DC4 (CTRL-T)
0010101	21	15	NAK (CTRL-U)
0010110	22	16	SYN (CTRL-V)
0010111	23	17	ETB (CTRL-W)
0011000	24	18	CAN (CTRL-X)
0011001	25	19	EM (CTRL-Y)
0011010	26	1A	SUB (CTRL-Z)
0011011	27	1B	ESC (CTRL-[)
0011100	28	1C	FS (CTRL-\)
0011101	29	1D	GS (CTRL-])
0011110	30	1E	RS (CTRL-^)
0011111	31	1F	US (CTRL-_)
0100000	32	20	(SPACE)

Table C-2. (continued)

Binary	Decimal	Hexadecimal	
0100001	33	21	!
0100010	34	22	"
0100011	35	23	#
0100100	36	24	$
0100101	37	25	%
0100110	38	26	&
0100111	39	27	'
0101000	40	28	(
0101001	41	29	)
0101010	42	2A	*
0101011	43	2B	+
0101100	44	2C	,
0101101	45	2D	-
0101110	46	2E	.
0101111	47	2F	/
0110000	48	30	0
0110001	49	31	1
0110010	50	32	2
0110011	51	33	3
0110100	52	34	4
0110101	53	35	5
0110110	54	36	6
0110111	55	37	7
0111000	56	38	8
0111001	57	39	9
0111010	58	3A	:
0111011	59	3B	;
0111100	60	3C	<
0111101	61	3D	=
0111110	62	3E	>
0111111	63	3F	?
1000000	64	40	@

Table C-2. (continued)

Binary	Decimal	Hexadecimal
1000001	65	41	A
1000010	66	42	B
1000011	67	43	C
1000100	68	44	D
1000101	69	45	E
1000110	70	46	F
1000111	71	47	G
1001000	72	48	H
1001001	73	49	I
1001010	74	4A	J
1001011	75	4B	K
1001100	76	4C	L
1001101	77	4D	M
1001110	78	4E	N
1001111	79	4F	O
1010000	80	50	P
1010001	81	51	Q
1010010	82	52	R
1010011	83	53	S
1010100	84	54	T
1010101	85	55	U
1010110	86	56	V
1010111	87	57	W
1011000	88	58	X
1011001	89	59	Y
1011010	90	5A	Z
1011011	91	5B	[
1011100	92	5C	\
1011101	93	5D	]
1011110	94	5E	^
1011111	95	5F	<
1100000	96	60	`

Table C-2. (continued)

Binary	Decimal	Hexadecimal	
1100001	97	61	a
1100010	98	62	b
1100011	99	63	c
1100100	100	64	d
1100101	101	65	e
1100110	102	66	f
1100111	103	67	g
1101000	104	68	h
1101001	105	69	i
1101010	106	6A	j
1101011	107	6B	k
1101100	108	6C	l
1101101	109	6D	m
1101110	110	6E	n
1101111	111	6F	o
1110000	112	70	p
1110001	113	71	q
1110010	114	72	r
1110011	115	73	s
1110100	116	74	t
1110101	117	75	u
1110110	118	76	v
1110111	119	77	w
1111000	120	78	x
1111001	121	79	y
1111010	122	7A	z
1111011	123	7B	{
1111100	124	7C	|
1111101	125	7D	}
1111110	126	7E	~
1111111	127	7F	DEL

End of Appendix C

ERROR MESSAGES

This appendix lists the error messages returned by the internal components of GEM DOS and by the GEM DOS programmer's utilities. subsections are arranged alphabetically by the name of the internal component or utility. Each subsection has error messages listed alphabetically, with explanations and suggested user responses.

D.1. AS68 ERROR MESSAGES

The GEM DOS assembler, AS68, returns both nonfatal, diagnostic error messages and fatal error messages. Fatal errors stop the assembly of your program. There are two types of fatal errors: user-recoverable fatal errors and fatal errors in the internal logic of AS68.

D.1.1. AS68 Diagnostic Error Messages

Diagnostic messages report errors in the syntax and context of the program being assembled without interrupting assembly. Refer to the Motorola 16-Bit Microprocessor User's Manual for a full discussion of the assembly language syntax.

Diagnostic error messages appear in the following format:

& line no. error message text

The ampersand (&) indicates that the message comes from AS68. The line no. indicates the line in the source code where the error occurred. The error message text describes the error. Diagnostic error messages appear at the console after assembly, followed by a message indicating the total number of errors. In a print-out, they print on the line preceding the error. Table D-1 lists the AS68 diagnostic error messages in alphabetical order.

Table D-1. AS68 Diagnostic Error Messages

Message	Meaning
& line no. backward assignment to *	The assignment statement in the line indicated illegally assigns the location counter (*) backward. Change the location counter to a forward assignment and reassemble the source file.

Table D-1. (continued)

Message Meaning

& line no. bad use of symbol
A symbol in the source line indicated has been defined as both global and common. A symbol can be either global or common, but not both. Delete one of the directives and reassemble the source file.

& line no. constant required
An expression on the line indicated requires a constant. Supply a constant and reassemble the source file.

& line no. end statement not at end of source
The end statement must be at the end of the source code. The end statement cannot be followed by a comment or more than one carriage return. Place the end statement at the end of the source code, followed only by a single carriage return, and reassemble the source file.

& line no. illegal addressing mode
The instruction on the line indicated has an invalid addressing mode. Provide a valid addressing mode and reassemble the source file.

& line no. illegal constant
The line indicated contains an illegal constant. Supply a valid constant and reassemble the source file.

& line no. illegal expr
The line indicated contains an illegal expression. Correct the expression and reassemble the source file.

& line no. illegal external
The line indicated illegally contains an external reference to an 8-bit quantity. Rewrite the source code to define the reference locally or use a 16-bit reference and reassemble the source file.

& line no. illegal format
An expression or instruction in the line indicated is illegally formatted. Examine the line. Reformat where necessary and reassemble the source file.

& line no. illegal index register
The line indicated contains an invalid index register. Supply a valid register and reassemble the source file.

Table D-1. (continued)

Message Meaning

& line no. illegal relative address
An addressing mode specified is not valid for the instruction in the line indicated. Refer to the Motorola 16-Bit Microprocessor User's Manual for valid register modes for the specified instruction. Rewrite the source code to use a valid mode and reassemble the file.

& line no. illegal shift count
The instruction in the line indicated shifts a quantity more than 31 times. Modify the source code to correct the error and reassemble the source file.

& line no. illegal size
The instruction in the line indicated requires one of the following three size specifications: b (byte), w (word), or l (longword). Supply the correct size specification and reassemble the source file.

& line no. illegal string
The line indicated contains an illegal string. Examine the line. Correct the string and reassemble the source file.

& line no. illegal text delimiter
The text delimiter in the line indicated is in the wrong format. Use single quotes ('text') or double quotes ("text") to delimit the text and reassemble the source file.

& line no. illegal 8-bit displacement
The line indicated illegally contains a displacement larger than 8-bits. Modify the code and reassemble the source file.

& line no. illegal 8-bit immediate
The line indicated illegally contains an immediate operand larger than 8-bits. Use the 16- or 32-bit form of the instruction and reassemble the source file.

& line no. illegal 16-bit displacement
The line indicated illegally contains a displacement larger than 16-bits. Modify the code and reassemble the source file.

& line no. illegal 16-bit immediate
The line indicated illegally contains an immediate operand larger than 16-bits. Use the 32-bit form of the instruction and reassemble the source file.

Table D-1. (continued)

Message Meaning

& line no. invalid data list
One or more entries in the data list in the line indicated is invalid. Examine the line for the invalid entry. Replace it with a valid entry and reassemble the source file.

& line no. invalid first operand
The first operand in an expression in the line indicated is invalid. Supply a valid operand and reassemble the source file.

& line no. invalid instruction length
The instruction in the line indicated requires one of the following three size specifications: b (byte), w (word), or l (longword). Supply the correct size specification and reassemble the source file.

& line no. invalid label
A required operand is not present in the line indicated, or a label reference in the line is not in the correct format. Supply a valid label and reassemble the source file.

& line no. invalid opcode
The opcode in the line indicated is non-existent or invalid. Supply a valid opcode and reassemble the source file.

& line no. invalid second operand
The second operand in an expression in the line indicated is invalid. Supply a valid operand and reassemble the source file.

& line no. label redefined
This message indicates that a label has been defined twice. The second definition occurs in the line indicated. Rewrite the source code to specify a unique label for each definition and reassemble the source file.

& line no. missing)
An expression in the line indicated is missing a right parenthesis. Supply the missing parenthesis and reassemble the source file.

& line no. no label for operand
An operand in the line indicated is missing a label. Supply a label and reassemble the source file.

Table D-1. (continued)

Message Meaning

& line no. opcode redefined
A label in the line indicated has the same mnemonics as a previously specified opcode. Respecify the label so that it does not have the same spelling as the mnemonic for the opcode. Reassemble the source file.

& line no. register required
The instruction in the line indicated requires either a source or destination register. Supply the appropriate register and reassemble the source file.

& line no. relocation error
An expression in the line indicated contains more than one externally defined global symbol. Rewrite the source code. Either make one of the externally defined global symbols a local symbol, or evaluate the expression within the code. Reassemble the source file.

& line no. symbol required
A statement in the line indicated requires a symbol. Supply a valid symbol and reassemble the source file.

& line no. undefined symbol in equate
One of the symbols in the equate directive in the line indicated is undefined. Define the symbol and reassemble the source file.

& line no. undefined symbol
The line indicated contains an undefined symbol that has not been declared global. Either define the symbol within the module or define it as a global symbol and reassemble the source file.

D.1.2. User-recoverable Fatal Error Messages

Table D-2 describes fatal error messages for AS68. When an error occurs because the disk is full, AS68 creates a partial file. Erase the partial file to ensure that you do not try to link it.

Table D-2. AS68 User-recoverable Fatal Error Messages

Message	Meaning
& cannot create init: AS68SYMB.DAT	AS68 cannot create the initialization file because the path name is incorrect or the disk to which it was writing the file is full. If you used the -S switch to redirect the symbol table to another disk, check the path name. If it is correct, the disk is full. Erase unnecessary files, if any, or insert a new disk before you reinitialize AS68. Erase the partial file that was created on the full disk to ensure that you do not try to link it.
& expr opstk overflow	An expression in the line indicated contains too many operations for the operations stack. Simplify the expression before you reassemble the source code.
& expr tree overflow	The expression tree does not have space for the number of terms in one of the expressions in the indicated line of source code. Rewrite the expression to use fewer terms before you reassemble the source file.
& I/O error on loader output file	The disk to which AS68 was writing the loader output file is full. AS68 wrote a partial file. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.
& I/O write error on it file	The disk to which AS68 was writing the intermediate text file is full. AS68 wrote a partial file. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.
& it read error itoffset= no.	The disk to which AS68 was writing the intermediate text file is full. AS68 wrote a partial file. The variable itoffset= no. indicates the first zero-relative byte number not read. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.

Table D-2. (continued)

Message	Meaning
&	Object file write error The disk to which AS68 was writing the object file is full. AS68 wrote a partial file. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.
&	overflow of external table The source code uses too many externally defined global symbols for the size of the external symbol table. Eliminate some externally defined global symbols and reassemble the source file.
&	Read Error On Intermediate File: ASXXXXn The disk to which AS68 was writing the intermediate text file ASXXXX is full. AS68 wrote a partial file. The variable n indicates the drive on which ASXXXX is located. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.
&	symbol table overflow The program uses too many symbols for the symbol table. Eliminate some symbols before you reassemble the source code.
&	Unable to open file filename The source filename indicated by the variable filename is invalid or has an invalid path name. Check the path name and the filename. Respecify the command line before you reassemble the source file.
&	Unable to open input file The filename in the command line indicated does not exist or has an invalid path name. Check the path name and the filename. Respecify the command line before you reassemble the source file.
&	Unable to open temporary file You used an invalid path name or the disk to which AS68 was writing is full. Check the path name. If it is correct, the disk is full. Erase unnecessary files, if any, or insert a new disk before you reassemble the source file.

Table D-2. (continued)

Message	Meaning
&	Unable to read init file: AS68SYMB.DAT The path name used to specify the initialization file is invalid or the assembler has not been initialized. Check the path name. Respecify the command line before you reassemble the source file. If the assembler has not been initialized, refer to Section 5 for instructions.
&	Write error on init file: AS68SYMB.DAT The disk to which AS68 was writing the initialization file is full. AS68 wrote a partial file. Erase unnecessary files, if any, or insert a new disk and reassemble the source file. Erase the partial file that was created on the full disk to ensure that you do not try to link it.
&	write error on it file The disk to which AS68 was writing the intermediate text is full. AS68 wrote a partial file. Erase unnecessary files, if any, or insert a new disk. Erase the partial file that was created on the full disk to ensure that you do not try to link it. Reassemble the source file.

D.1.3. AS68 Internal Logic Error Messages

The following are messages indicating fatal errors in the internal logic of AS68:

```
&  doitrd:  buffer botch pitix=nnn itbuf=nnn end=nnn
&  doitwr: it buffer botch
&  invalid radix in oconst
&  i.t. overflow
&  it sync error itty=nnn
&  seek error on it file
&  outword: bad rlflg
```

If you receive one of these messages, contact your vendor for assistance. You should do the following:

* Indicate which version of the operating system you are using.

* Describe your system's hardware configuration.

* Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.

D.2. LO68 ERROR MESSAGES

The GEM DOS Linker, LO68, returns two types of fatal error messages: diagnostic and logic. Both types of fatal error messages have the following format:

```
: error message text
```

The colon (:) indicates that the error message comes from LO68. The "error message text" describes the error.

D.2.1. Fatal Diagnostic Error Messages

A fatal diagnostic error prevents your program from linking. When the error is caused by a full disk, erase the partial file that LO68 created on the disk that received the error to ensure that you do not use the file. The LO68 diagnostic errors are listed in Table D-3 in alphabetic order with explanations and suggested user responses.

Table D-3. LO68 Fatal Diagnostic Error Messages

Message	Meaning
: duplicate definition in p,filename	The symbol indicated by the variable p is defined twice. The variable filename indicates the file in which the second definition occurred. Rewrite the source code. Provide a unique definition for each symbol and reassemble or recompile the source code before you relink the file.
: file format error: filename	The file indicated by the variable filename is either not an object file or the file has been corrupted. Ensure that the file is an object file, output by the assembler or compiler. Reassemble or recompile the file before you relink it.
: File Format Error: Invalid symbol flags = flags	LO68 does not recognize the symbol flags indicated by the variable flags. The file LO68 read is either not an object file or has been corrupted. Ensure that the file is an object file, output by the assembler or compiler. Reassemble or recompile the file before you relink it.
: File Format Error: invalid relocation flag in filename	The contents of the file indicated by the variable filename are incorrectly formatted. The file either is not an object file or has been corrupted. Ensure that the file is an object file, output by the assembler or compiler. If the file is an object file but this error occurs, the file has been corrupted. Reassemble or recompile the file before you relink it.

Table D-3. (continued)

Message Meaning

: File Format Error: no relocation bits in filename
The file indicated by the variable filename either is not an object file or has been corrupted. Ensure that the file is an object file, output by the assembler or compiler. If the file is an object file but this error occurs, then the file has been corrupted. Reassemble or recompile the file before you relink it.

: Illegal option p
The option in the command line indicated by the variable p is invalid. Supply a valid option and relink.

: Invalid lo68 argument list
This message indicates format errors or invalid options in the command line. Examine the command line to locate the error. Correct the error and relink.

: output file write error
The disk to which LO68 is writing is full. Erase unnecessary files, if any, or insert a new disk before you reenter the LO68 command line.

: read error on file: filename
The object file indicated by the variable filename does not have enough bytes. The file either is incorrectly formatted or has been corrupted. This error is commonly caused when the input to LO68 is a partially assembled or compiled object file. The assembler, AS68, and some compilers create partial object files when they receive the disk full abort message while assembling or compiling a file. Ensure that the file is a complete object file. Reassemble or recompile the file before you relink it.

: symbol table overflow
The object code contains too many symbols for the size of the symbol table. Rewrite the source code to use fewer symbols. Reassemble or recompile the source code before you relink the file.

: Unable to create pathname
The output file indicated by pathname has an invalid path name, or the disk to which LO68 is writing is full. Check the path name. If it is correct, the disk is full. Erase unnecessary files, if any, or insert a new disk before you reenter the LO68 command line.

Table D-3. (continued)

Message Meaning

: unable to open filename
The filename indicated by the variable filename is invalid or the file does not exist. Check the filename before you reenter the LO68 command line.

: Unable to open temporary file: pathname
Either the file, indicated by pathname, has an invalid drive code in the path name, specified by the f option, or the disk to which LO68 is writing is full. Check the path name. If it is correct, the disk is full. Erase unnecessary files, if any, or insert a new disk before you reenter the LO68 command line.

: Undefined symbol(s)
The symbol or symbols which are listed one per line following the error message are undefined. Provide a valid definition and reassemble the source code before you reenter the LO68 command line.

D.2.2. LO68 Internal Logic Error Messages

This section lists messages indicating fatal errors in the internal logic of LO68. If you receive one of these messages, contact the place you purchased your system for assistance. You should do the following:

* Indicate which version of the operating system you are using.

* Describe your system's hardware configuration.

* Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.

Errors:

: asgnext botch

: finalwr: text size error

: relative address overflow at lx in sn

: seek error on file filename

: short address overflow in filename

: unable to reopen filename

D.3. AR68 ERROR MESSAGES

The GEM DOS Archive utility, AR68, returns two types of fatal error messages: diagnostic and logic. Both types of fatal error messages show at the console as they occur.

D.3.1. Fatal Diagnostic Error Messages

Table D-4 lists AR68 fatal error messages in alphabetical order with explanations and suggested user responses.

Table D-4. AR68 Fatal Diagnostic Error Messages

Message	Meaning
filename not in archive file	The object module indicated by the variable filename is not in the library. Check the filename before you reenter the command line.
cannot create filename	The path name for the file indicated by the variable filename is invalid, or the disk to which AR68 is writing is full. Check the path name. If it is valid, the disk is full. Erase unnecessary files, if any, or insert a new disk before you reenter the command line.
cannot open filename	The file indicated by the variable filename cannot be opened because the filename or the path name is incorrect. Check the path name and the filename before you reenter the command line.
invalid option flag: x	The symbol, letter, or number in the command line indicated by the variable x is an invalid option. Refer Section 3 of this manual for an explanation of the AR68 command line options. Specify a valid option and reenter the command line.
not archive format: filename	The file indicated by the variable filename is not a library. Ensure that you are using the correct filename before you reenter the command line.
not object file: filename	The file indicated by the variable filename is not an object file, and cannot be added to the library. Any file added to the library must be an object file, output by the assembler, AS68, or the compiler. Assemble or compile the file before you reenter the AR68 command line.

Table D-4. (continued)

Message Meaning

one and only one of DRTWX flags required

The AR68 command line requires one of the D, R, T, W, or X commands, but not more than one. Reenter the command line with the correct command. Refer to Section 7 for an explanation of the AR68 commands.

filename not in library

The object module indicated by the variable filename is not in the library. Ensure that you are requesting the filename of an existing object module before you reenter the command line.

Read error on filename

The file indicated by the variable filename cannot be read. This message means one of three things: the file listed at filename is corrupted; a hardware error has occurred; or when the file was created, it was not correctly written by AR68 due to an error in the internal logic of AR68.

Cold start the system and retry the operation. If you receive this error message again, you must erase and recreate the file. Use your backup file, if you maintained one. If the error reoccurs, check for a hardware error. If the error persists, contact the the vendor of your system for assistance. You should do the following:

* Indicate which version of the operating system you are using.

* Describe your system's hardware configuration.

* Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.

Table D-4. (continued)

Message	Meaning
temp file write error	The disk to which AR68 was writing the temporary file is full. Erase unnecessary files, if any, or insert a new disk before you reenter the command line.
	usage: AR68 DRTWX[AV][F D:] [OPMOD] ARCHIVE OBMOD1 [OBMOD2...][>filespec]
	This message indicates a syntax error in the command line. The correct format for the command line is given, with the possible options in brackets. Refer to Section 7 for a more detailed explanation of the command line.
Write error on filename	The disk to which AR68 is writing the file indicated by the variable filename is full. Erase unnecessary files, if any, or insert a new disk before you reenter the command line.

D.3.2. AR68 Internal Logic Error Messages

The following are messages that indicate fatal errors in the internal logic of AR68:

cannot reopen filename

seek error on library

Seek error on tempname

Unable to recreate--library is in filename

If you receive any of these messages except the last, contact your vendor for assistance. You should do the following:

* Indicate which version of the operating system you are using.

* Describe your system's hardware configuration.

* Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.

For the last error, Unable to recreate--library is in filename, you should rename the temporary file indicated by the variable filename. AR68 used the library to create the temporary file, then deleted the library in order to replace it with the updated temporary file. This error occurred because AR68 cannot write the temporary file back to the original location. The entire library is in the temporary file.

D.4. DUMP ERROR MESSAGES

DUMP returns fatal, diagnostic error messages at the console. Table D-5 lists the DUMP error messages in alphabetical order with explanations and suggested user responses.

Table D-5. DUMP Error Messages

Message	Meaning
Unable to open filename	Either the path name for the input file indicated by the variable filename is incorrect, or the filename is misspelled. Check the filename and path name before you reenter the DUMP command line.
Usage: dump [-shhhhhhh] file	The command line syntax is incorrect. The correct syntax is given in the error message. Specify the DUMP command and the filename. If you want to display the contents of the file from a specific address in the file, specify the -S option followed by the address. Refer to Section 8.2 for discussion of the DUMP command line and options.

D.5. SIZE68 ERROR MESSAGES

SIZE68 returns fatal, diagnostic error messages at the console. Table D-6 lists the SIZE68 error messages in alphabetical order with explanations and suggested user responses.

Table D-6. SIZE68 Error Messages

Message	Meaning
File format error: filename	The file indicated by the variable filename is neither an object file nor a command file. SIZE68 requires either an object file, output by the assembler or the compiler, or a command file, output by the linker. Ensure that the file specified is one of these and reenter the SIZE68 command line.
read error on filename	The file indicated by the variable filename is truncated. Rebuild the file. Reassemble or recompile, then relink the source file before you reenter the SIZE68 command line.
unable to open filename	Either the path name is incorrect, or the file indicated by the variable filename does not exist. Check the path name and filename. Reenter the SIZE68 command line.

D.6. SENDC68 ERROR MESSAGES

SENDC68 returns two types of fatal error messages: diagnostic and internal logic error messages.

D.6.1. Diagnostic Error Messages

Table D-7 lists the SENDC68 diagnostic error messages in alphabetical order with explanations and suggested user responses.

Table D-7. SENDC68 Diagnostic Error Messages

Message	Meaning
file format error: filename	The file indicated by the variable filename is not a command file. The file input to SENDC68 must be a command file, output by the linker (LO68). Ensure that the file specified is a command file.
read error on file: filename	The file indicated by the variable filename is truncated. Rebuild the file by recompiling or reassembling, and relink it before you reenter the SENDC68 command line.
unable to create filename	This message indicates an invalid path name for the output file indicated by the variable filename. It can also mean that the disk to which SENDC68 is writing is full. Check the path name. If it is correct, the disk is full. Erase unnecessary files, if any, or insert a new disk before you reenter the SENDC68 command line.
unable to open filename	The input file indicated by the variable filename does not exist. Check the filename and retype the SENDC68 command line.
Usage: sendc68 [-] commandfile [outputfile]	This message indicates a syntax error in the SENDC68 command line. The correct syntax is given in the error message. Retype the command line using the correct syntax.

D.6.2. SENDC68 Internal Logic Error Messages

The following is a message that indicates a fatal error in the internal logic of SENDC68:

seek error on file filename

If you receive this message, contact your vendor. You should do the following:

* Indicate which version of the operating system you are using.

* Describe your system's hardware configuration.

* Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.

D.7. SID68 ERROR MESSAGES

The GEM DOS debugger, SID68, returns two types of error messages: nonfatal diagnostic error messages, and messages indicating fatal errors in the internal logic of SID68.

D.7.1. Diagnostic Error Messages

Diagnostic error messages are returned at the console as the errors occur. Table D-8 lists the SID68 error messages in alphabetical order with explanations and suggested user responses.

Table D-8. SID68 Diagnostic Error Messages

Message — Meaning

Bad or non-existent RAM at HEX no.

This error occurs in response to a Set (S), Set Word (SW), or Set Longword (SL) command. The message indicates one of two things:

* The memory location at HEX no. is Read-Only, an I/O port, or non-existent. Use another location.

* The memory location is damaged. Check the hardware.

Bad relocation bits

This message is returned from the BDOS Program Load Function (59), and means one of two things:

* The command file has been corrupted. Rebuild the file. Reassemble or recompile the source file, then relink the file before you reenter the SID68 command line.

* The file is linked to an absolute location in memory that is already occupied by SID68. Link the file to another location.

  SID68 occupies approximately 20K of memory, and resides at the highest addresses within the TPA. The recommended location for linking your file is the base address of the TPA + 100H. BDOS Function 63, Get/Set TPA Limits, returns the high and low boundaries of the TPA.

Table D-8. (continued)

Message Meaning

Cannot create file

This error occurs during a Write (W) command. The disk to which SID68 is writing has no more directory space available; in effect, the disk is full. If you have another drive available, reenter the W command and direct the file to the disk on that drive. If you do not have another drive available, you must exit SID68 (and lose the contents of memory). Erase unnecessary files, if any, or insert a new disk.

Cannot open file

This error occurs during a R command. It indicates an incorrect path name or filename. Check the path name and filename before you reenter the command line.

Cannot open program file

This message occurs in response to an E command. It indicates an incorrect path name or filename. Check the path name and filename before you reenter the command line.

ERROR, no program or file loaded

This error message occurs in response to a V command when you specify the command but no file is loaded. Load a file before you reenter the V command. The file can be loaded with an E or R command, or by specifying the filename when you invoke SID68.

File too big -- read truncated

This message occurs during a R command when the file being read is too large to fit in memory. SID68 reads only the portion of the file that can be read into the existing memory. To debug this program, additional memory boards must be added to the system configuration.

File write error

The disk to which SID68 is writing is full or the disk contains a bad sector. Retry the command. If the error persists, and you have another disk drive available, redirect the output to the disk on that drive. If you do not have another drive available, you must exit SID68. Use the STAT command to check the space on the disk. If it is full, erase unnecessary files, if any, or insert a new disk. Because the contents of memory are lost when you exit SID68, you must reload the file in memory. If the disk was not full, it has a bad sector. You should replace the disk.

Table D-8. (continued)

Message Meaning

**illegal size field
This error occurs during an L command. The size field of the instruction being disassembled has an illegal value. Use a D command to display the location of the error. This error could be caused by one of three things:

* The memory location being disassembled does not contain an instruction. Ensure that the area selected is an instruction. If not, reenter the L command with a correct location.

* The size field of the instruction has been corrupted. Use the debugging commands in SID68 to look for an error that causes the program to overwrite itself. Refer to Section 9 for a complete description of the SID68 commands and options.

* An invalid instruction was generated by the compiler or assembler used to create the program. Recompile or reassemble the source file before you reinvoke SID68.

Insufficient memory or bad file header
This message occurs in response to an E command. The error could be caused by one of three things:

* The system you are using does not have enough memory available. Ensure that the program and SID68 fit into the TPA. Exit SID68. Use the SIZE68 utility to display the amount of space your program occupies in memory. SID68 is approximately 20K bytes. The BDOS Get/Set TPA Limits Function (63) returns the high and low boundaries of the TPA. If you do not have sufficient space in the TPA to execute your command file and SID68 simultaneously, additional memory boards must be added to your system configuration.

Table D-8. (continued)

Message Meaning

* The file is not a command file or has a corrupted header. If the command file does not run, but you are sure that your memory space is adequate, use the R command to look at the file and check the format. You might be trying to debug a file that is not a command file. If it is a command file, the header might have been corrupted. Reassemble or recompile the source file before you reenter the E command line. If the error persists, it might be caused by an error in the internal logic of SID68. Contact your vendor for assistance. You should do the following:

   -- Indicate which version of the operating system you are using.

   -- Describe your system's hardware configuration.

   -- Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.

* The command file you are debugging is linked to an absolute location in memory that is already occupied by SID68. SID68 is approximately 20K bytes, and usually resides in the highest addresses of the TPA. The recommended location for linking your file is the base address of the TPA + 100H. The BDOS Get/Set TPA Limits Function (63) returns the high and low boundaries of the TPA.

Table D-8. (continued)

Message Meaning

Read error

This message indicates one of three things. Try the operation again. If the error persists, you have one of the following problems:

* A write error at the time the file was created. You must recreate the file. If the error reoccurs, or if you cannot write to the disk, the disk is bad.

* A bad disk. Use PIP or COPY to copy the file from the bad disk to a new disk. Any files that cannot be copied must be recreated or replaced from backup files. Discard the damaged disk.

* A hardware error. If the error persists, check your hardware.

unknown opcode

This error occurs in response to a L command if the memory location being disassembled does not contain a valid instruction. The error might have been caused by one of three things:

* You gave the L command the wrong address. Reenter the L command with the correct address.

* The file is not a command file. Make sure that the file you specify is a command file and reenter the L command.

* The command file has been corrupted. Reassemble or recompile the source file before you reread it into memory with a Load for Execution (E) or Read (R) command, as appropriate. If the problem persists, use the debugging commands in SID68 to look for an error in the program that causes it to overwrite itself. Refer to the Section 9 for a complete description of the SID68 commands and options.

D.7.2. SID68 Internal Logic Error Messages

The following are messages indicating fatal errors in the internal logic of SID68:

illegal instruction format #

Unknown program load error

If you receive one of these messages, contact your vendor for assistance. You should do the following:

* Indicate which version of the operating system you are using.

* Describe your system's hardware configuration.

* Provide sufficient information to reproduce the error. Indicate which program was running at the time the error occurred. If possible, you should also provide a disk with a copy of the program.

End of Appendix D

MOTOROLA S-RECORD FORMAT

The Motorola S-record format is a method of representing binary memory images in an ASCII form. The primary use of S-records is to provide a convenient form for transporting programs between computers. Since most computers have a means of reading and writing ASCII information, the format is widely applicable. The SENDC68 utility provided with GEM DOS can be used to convert programs into S-record form.

An S-record file consists of a sequence of S-records of various types. The entire content of an S-record is ASCII. When a hexadecimal number needs to be represented in an S-record, it is represented by the ASCII characters for the hexadecimal digits comprising the number. Each S-record contains the five fields shown in Figure E-1. Table E-1 describes each field.

Field:	S	type	length	address	data	cksum
Size*:	1	1	2	2, 4 or 6	varies	2

* in characters

Figure E-1. S-record Format

Table E-1. S-record Field Descriptions

Field	Description
S	The ASCII character S. This signals the beginning of the S-record.
type	A digit between 0 and 9, represented in ASCII, with the exceptions that 4 and 6 are not allowed. The use of each type value is explained in Table E-2.
length	The number of character pairs in the record, excluding the first three fields. (That is, one half the number of total characters in the address, data, and cksum fields.) This field has two hexadecimal digits, representing a one-byte quantity.
address	The address at which the data portion of the record is to reside in memory. The data goes to this address and successively higher numbered addresses. The length of this field is determined by the record type in the second byte of the S-record.
data	A variable length field containing the actual data to be loaded into memory. Specify each byte of data as a pair of hexadecimal digits in ASCII.
cksum	A checksum compute over the length, address, and data fields. Compute the checksum as follows: * add the values of the character pairs in the length, address, and data fields * take the one's complement of the sum * drop the most significant byte Enter the least significant byte value in the cksum field as two ASCII hexadecimal digits.

There are eight types of S-records. They can be divided into two categories: records containing actual data and records used to define and delimit groups of data-containing records. Types 1, 2, and 3 are reserved for records in the first category; types 0, 5, 7, 8, and 9 for reserved for records in the second category. Types 4 and 6 are not allowed. Table E-2 defines the types.

Note: All byte values in Table E-2 are expressed as two ASCII characters representing the hexadecimal value.

Table E-2. S-Record Type Definitions

Type	Meaning
0	This type is a header record used at the beginning of a group of S-records. The data field can contain any desired identifying information. The address field is two bytes long and is normally zero.
1	This type of record contains normal data. The address field is two bytes long.
2	This type is the same as type 1 except the address field is 3 bytes long.
3	This type is the same as type 1 except the address field is 4 bytes long.
5	This type indicates the number of type 1, 2, and 3 records in a group of S-records. The count is placed in the address field. The data field is empty.
7	This record signals the end of a block of type 3 S-records. If desired, the 4-byte address field can be used to contain an address at which to pass control. The data field is empty.
8	This type is the same as type 7 except that it ends a block of type 2 S-records and the address field is 3 bytes long.
9	This type is the same as type 7 except that it ends a block of type 1 S-records and the address field is 2 bytes long.

End of Appendix E